Technology
Arts Sciences
TH Köln

Erweiterung eines Lernmoduls für Boolesche Algebra

Bachelorarbeit

ausgearbeitet von

Sergej Görzen

zur Erlangung des akademischen Grades

Bachelor of Science (B.Sc.)

vorgelegt an der

Technischen Hochschule Köln
Campus Gummersbach
Fakultät für Informatik und
Ingenieurwissenschaften

im Studiengang

Allgemeine Informatik

Erster Prüfer: Prof. Dr.-Ing. Martin Eisemann
Technische Hochschule Köln

Zweiter Prüfer: Prof. Dr. Christian Kohls
Technische Hochschule Köln

Gummersbach, im Februar 2017

Adressen: Sergej Görzen
Hähner Weg 6
51580 Reichshof
sergej-goerzen@web.de

Prof. Dr.-Ing. Martin Eisemann
Technische Hochschule Köln
Institut für Informatik
Steinmüllerallee 1
51643 Gummersbach
martin.eisemann@th-koeln.de

Prof. Dr. Christian Kohls
Technische Hochschule Köln
Institut für Informatik
Steinmüllerallee 1
51643 Gummersbach
christian.kohls@th-koeln.de

Kurzfassung

Studenten nutzen immer mehr interaktive Lernangebote. Von Kommunikations- und Austauschplattformen bis hin zu Applikationen gibt es einen breiten, ständig wachsenden Informationszugang, der immer größer wird. Besonders seitdem mobile Endgeräte auf dem Markt sind, hat der Mensch fortwährenden Zugriff auf das Internet, wodurch cloudbasierte Lösungen immer interessanter werden. Diese können ohne Installation aufgerufen und genutzt werden. Dadurch müssen Studenten nicht immer ihren eigenen Laptop dabei haben, sondern können Computer an der Hochschule oder anderen Einrichtungen benutzen oder zu ihrem Smartphone greifen.
Interaktive Lernapplikationen bieten Interessenten die Möglichkeit effektiv unterwegs zu lernen. Besonders Logikaufgaben können solche Applikationen einen größeren Lern- und Spaßfaktor bieten.
In dieser Arbeit wird eine webbasierte Lernapplikation für Boolesche Algebra weiterentwickelt und für mobile Endgeräte optimiert. Zum Einsatz kommen HTML5, CSS3, JavaScript und etliche moderne Frameworks.
Die aktuelle Webapplikation, wird um ein KV Diagramm erweitert. Während der Dokumentation wird auf unterschiedliche Algorithmen und Strukturen eingegangen. Außerdem wurden Modelle für die wichtigsten Abläufe gestaltet.

Inhaltsverzeichnis

Kurzfassung **3**

Abbildungsverzeichnis **5**

Tabellenverzeichnis **6**

1 Einleitung **6**
1.1 Ziel der Arbeit . . . 6
1.2 Vorgehensweise . . . 7

2 Vorstellung vorhandener Apps **8**
2.1 Android . . . 8
2.1.1 Boole . . . 8
2.1.2 KVD - Karnaugh-Veitch-Diagramm . . . 9
2.2 Browser . . . 10
2.2.1 Boolesche Ausdrücke vereinfachen . . . 10
2.2.2 Karnaugh-Veitch Map . . . 11

3 Vorstellung der vorhandenen Lernplattform **13**
3.1 Lernplattform . . . 13
3.1.1 Theoretische Informatik . . . 13
3.1.2 Computergrafik und Animation . . . 14
3.2 Lernmodul für Boolesche Algebra . . . 14

4 Konzept **15**
4.1 Theorie . . . 15
4.1.1 Boolescher Ausdruck . . . 15
4.1.2 Wahrheitstabelle . . . 15
4.1.3 KV Diagramm . . . 16
4.1.4 Minimierung im KV Diagramm . . . 18
4.1.5 Schaltsysteme . . . 19
4.2 Designüberlegungen . . . 20
4.2.1 Mockups . . . 20
4.2.2 Anpassung der Mockups . . . 23
4.2.3 Systementwurf . . . 24
4.2.4 Optisches Design . . . 25
4.3 Überlegungen zur Datenstruktur des KV Diagramms . . . 25
4.4 Suche von größten Blöcken . . . 26
4.4.1 Rekursive Suche . . . 26
4.4.2 Iterative Bruteforce-Suche . . . 27

4.4.3 Iterative Suche durch Referenzen . . . 27
4.5 Umsetzungsentscheidung . . . 30

5 Anforderungsanalyse 31
5.1 Zielbestimmungen und Zielgruppen . . . 31
5.1.1 Produktperspektive . . . 31
5.1.2 Einsatzkontext . . . 31
5.1.3 Zielgruppen . . . 31
5.2 Funktionale Anforderungen . . . 32
5.3 Nicht-funktionale Anforderungen . . . 32
5.3.1 Funktionalität . . . 33
5.3.2 Zuverlässigkeit . . . 33
5.3.3 Benutzbarkeit . . . 34
5.3.4 Effizienz . . . 34
5.3.5 Änderbarkeit . . . 34
5.3.6 Übertragbarkeit . . . 34

6 Technischer Einsatzkontext 35
6.1 Überblick . . . 35
6.2 Entwicklungsumgebung und verwendete Software . . . 37
6.3 Komprimierung mit Hilfe von Gulp . . . 40
6.4 Gulp-Tasks innerhalb der WebStorm IDE . . . 41
6.5 Entscheidung für das EaselJS Framework . . . 43

7 Implementierung 44
7.1 Systemarchitektur . . . 44
7.1.1 Gesamtübersicht als Komponentendiagramm . . . 44
7.1.2 Komponentediagramm der Applikation . . . 45
7.1.3 Klassendiagramm . . . 46
7.1.4 Aktivitätsdiagramme . . . 50
7.2 Pseudocodes . . . 52
7.2.1 KV-Reflecting-Search-Algorithmus . . . 52
7.2.2 ColorRects-Algorithmus . . . 56
7.3 Überprüfen der Ergebnisse . . . 58

8 Vorstellung der Anwendung 59

9 Evaluation 63
9.1 Lauffähigkeit auf unterschiedlichen Plattformen . . . 63
9.1.1 Computer und Laptop . . . 63
9.1.2 Mobile Endgeräte . . . 66
9.2 Benutzerumfrage . . . 69

10 Selbstreflexion 70
10.1 Ablauf . . . 70
10.2 Schwierigkeiten . . . 70
10.2.1 Such-Algorithmus . . . 70

10.2.2 Optische Ungleichheiten 71
10.3 Fazit 71
10.4 Ausblick 72

Glossar **74**

Quellenverzeichnis **75**

Anhang **76**

Eidesstattliche Erklärung **78**

Abbildungsverzeichnis

2.1 Boole - Screenshots [GooglePlay (a)] 9
2.2 KVD - Screenshot [GooglePlay (b)] 10
2.3 Elektroniker Bude - Screenshotausschnitte [E-Bude] 11
2.4 Universität Marburg - Karnaugh-Veitch Map [Marburg] 12

3.1 Screenshot von Simulator for Boolean Algebra [eLearning Boolean Alg] . 14

4.1 Wahrheitstabelle für F=A∧B 16
4.2 KV Diagramm für F=A∧B 17
4.3 KV Diagramm für F=A∧B∨C 17
4.4 F=A∧(B∨¬B)∧C∨D Minimierung 18
4.5 Logikgatter 19
4.6 Schaltung von F=A∧B∨C 19
4.7 Mockup - Startansicht 20
4.8 Mockup - Wahrheitstabelle 21
4.9 Mockup - KV Diagramm 22
4.10 Mockup - Schaltsysteme 23
4.11 Mockup - KV Diagramm (bearbeitet) 24
4.12 KV Diagramm Netz 26
4.13 Iterativer Algorithmus für F=A∨B∨C∧D 28
4.14 Lösung für F=A∨B∨C∧D 28
4.15 F=A∧B∨C Algorithmusvorschau 29

6.1 Überblick vom technischen Einsatz 35
6.2 Beispiel für Useref 41
6.3 Eingabeaufforderung: gulp build 42
6.4 WebStorm Play-Button Konfiguration 42
6.5 WebStorm Play-Button - Run Dev 42

7.1 Komponentendiagramm für die Übersicht 45
7.2 Komponentendiagramm - AngularJS Module 46
7.3 Klassendiagramm der Applikation (gekürzt) 47
7.4 Aktivitätsdiagramm der Interaktion 50

7.5 Aktivitätsdiagramm für Farbebenenanalyse 51
7.6 Regextest von der Klammersuche [regex101] 58

8.1 Startansicht 59
8.2 Generierung eines KV Diagramms 60
8.3 Farbliche Blöcke und deren Textfelder 60
8.4 Überprüfung der Lösung 61
8.5 Lösung vom KV Diagramm anzeigen 61
8.6 Lösung der Eingaben anzeigen 62
8.7 Automatische Neuerstellung 62

9.1 Evaluation auf Windows mit Google Chrome 64
9.2 Evaluation auf Windows mit Firefox 64
9.3 Evaluation auf Mac OS X Yosemite mit Safari 65
9.4 Evaluation auf Android mit Google Chrome 1 66
9.5 Evaluation auf Android mit Google Chrome 2 66
9.6 Evaluation auf Android mit Firefox 1 67
9.7 Evaluation auf Android mit Firefox 2 67
9.8 Evaluation auf iPhone mit Safari 1 68
9.9 Evaluation auf iPhone mit Safari 2 68

Tabellenverzeichnis

6.1 Verwendete Programme 37
6.2 Backend Frameworks 38
6.3 Frontend Frameworks 40
6.4 Vergleich der HTML5 Canvas Frameworks 43

7.1 Klassen für den Suchalgorithmus 48
7.2 Klassen für die Datenstruktur 48
7.3 Klassen für die grafische Darstellung 49

10.1 Checkliste der funktionalen Anforderungen 71

1 Einleitung

E-Learning hat in der heutigen Zeit, besonders an Hochschulen, einen höheren Stellenwert gewonnen. Laut dem Stadtportal der Hauptstadt Berlin *BerlinOnline Stadtportal*[1] nutzen Studenten Internet-Plattformen, um auf Vorlesungskripte und -folien zuzugreifen, die von Dozenten zur Verfügung gestellt werden [dpa (2011)].
Eine Möglichkeit das Angebotsspektrum der Internet-Plattformen zu erweitern ist der Einsatz von Tools, mit denen dann ein Themengebiet interaktiv oder visuell vertieft werden kann. An der TH Köln entwickeln Studenten zum Beispiel Webapplikationen für unterschiedliche Gebiete der Informatik. Auf einigen von denen wird in Kapitel 3 nochmals eingegangen.

1.1 Ziel der Arbeit

Ziel dieser Arbeit ist die Erweiterung des Lernmoduls für Boolesche Algebra. Das Lernmodul wurde von mir im Rahmen des Praxisprojekts (s. Datenträger) mit dem Titel *Umsetzung eines Lernmoduls für Boolesche Algebra* an der TH Köln entwickelt. Dieses wird in Kapitel 3.2 vorgestellt und im Verlauf des Projekts um weitere Funktionen erweitert. Dabei werden Ideen, Konzeption und Umsetzungsfortschritt festgehalten. Die Funktionalitäten werden zusätzlich auf mehreren Browsern und Geräten geprüft. Inkompatible Funktionen werden dann bei Bedarf angepasst.
Das Aufstellen einer Wahrheitstabelle (s. Kapitel 4.1.2) sowie eines KV Diagramm (Karnaugh Veitch Diagramm, s. Kapitel 4.1.3) ist ein Routinevorgang und auf Papier teilweise sehr aufwändig. Da der Schwerpunkt nicht beim Aufstellen sondern beim Lösen dieser Konstrukte liegt und das Lernmodul den Aufbau übernimmt, könnte das Lernmodul ein gutes Hilfsmittel dafür sein, sich verschärft auf das Lösen der Konstrukte zu konzentrieren. Die Zeit, die sonst für den Aufbau verloren geht, kann dafür genutzt werden mehr Aufgaben zu üben.

[1] BerlinOnline Stadtportal: https://www.berlin.de/adressen/internetservice/berlinonline-stadtportal-575b64b4299a8001e856f018051d7228.html

1.2 Vorgehensweise

Um dem Leser einen besseren Überblick zu verschaffen, wird in diesem Abschnitt die Herangehensweise an das Thema erläutert.
Zuerst werden in Kapitel 2 ähnliche Apps und Webtools betrachtet, um ein wenig Inspiration und Ideen zu gewinnen. Danach wird die vorhandene Lernplattform in Kapitel 3 und deren Tools kurz vorgestellt, damit geklärt wird, um welchen Gesamtkontext es sich handelt.
Darauf folgt die Konzeption (in Kapitel 4), die in Theorie, Designüberlegungen, Überlegungen zur Datenstruktur des KV Diagramms, Suche von größten Blöcken und Umsetzungsentscheidung aufgeteilt ist.
Bei der Anforderungsanalyse (s. Kapitel 5) wird die Zielbestimmung erläutert. Zusätzlich werden die Zielgruppen, Funktionale und Nicht-funktionale Anforderungen bestimmt.
In Kapitel 6 werden die verwendete Soft- und Hardware vorgestellt.
Außerdem folgen die Implementierung (in Kapitel 7), dessen Ergebnis (Kapitel 8) und Evaluation (Kapitel 9).
Zum Schluss wird in Kapitel 10 Ablauf und Ergebnis reflektiert sowie Fazit gezogen.

2 Vorstellung vorhandener Apps

In diesem Kapitel wird das Ergebnis der Domänenrecherche präsentiert. Dadurch wird ein Überblick über die jeweils vorhandenen Funktionen geboten. Dies ermöglicht eine bessere Reflexion zu diesem Projekt, die in Kapitel 10 erfolgt. Bei der Recherche wurde der Fokus hauptsächlich auf die Funktionalität und das Vorhandensein der Wahrheitstabelle, des KV Diagramm und der Minimierung eines Booleschen Ausdrucks gelegt, da die zu umsetzende Applikation ebenfalls diese Funktionen nach Fertigstellung besitzen wird.
Zuerst werden in Kapitel 2.1 zwei Apps für Android Geräte und anschließend in Kapitel 2.2 zwei Applikationen im Browser vorgestellt.

2.1 Android

Da es im Google PlayStore[1] einige verschiedene Apps für Boolesche Algebra gibt und diese sich sehr ähnlich sind, werden nur die Apps vorgestellt, die meisten Funktionen sowie eine gute Bewertung (mindestens vier von fünf Sternen) haben.

2.1.1 Boole

Die *Boole*[2] App bietet insgesamt vier Grundfunktionen, die über Tabs aufrufbar sind: Generierung einer Wahrheitstabelle (*Truth Table*). Erzeugung eines KV Diagramms (*Karnaugh Map*), Darstellung in kanonischer Normalform (*Canonical Form*) und Umformung in duale Form (*Dual Function*).
In Abbildung 2.1 sind die Startansicht und zwei für die Recherche wichtige Funktionen zu sehen: die Wahrheitstabelle und das KV Diagramm. Ganz oben auf der Benutzeroberfläche der App stehen ein Eingabefeld und Button zum Bestätigen des eines Booleschen Ausdrucks zur Verfügung. Nach dem Bestätigen wird jede Ansicht der obengenannten Funktionen aktualisiert. Es kann zwischen den gewünschten Ansichten durch Wischen nach rechts oder links gewechselt werden.

[1] Google PlayStore: `https://play.google.com/store?hl=de`
[2] Boole: `https://play.google.com/store/apps/details?id=boole.riazavalverde.com&hl=de`

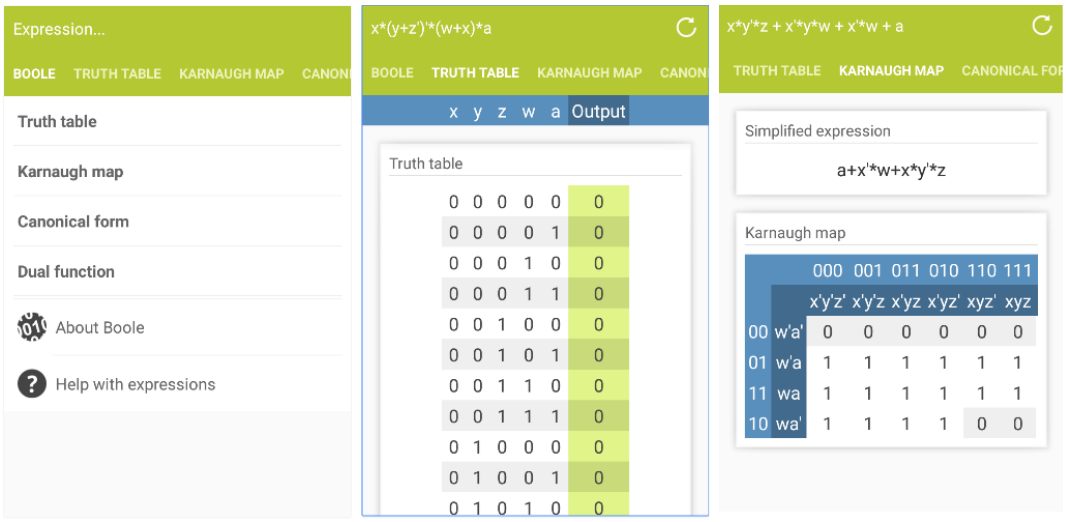


Abbildung 2.1: Boole - Screenshots [GooglePlay (a)]

2.1.2 KVD - Karnaugh-Veitch-Diagramm

Die *KVD - Karnaugh-Veitch-Diagramm*[3] App verzichtet auf die textuelle Eingabe eines Booleschen Ausdrucks. Stattdessen wird der Ausdruck durch das Betätigen einer Zelle innerhalb des KV Diagramms kontrolliert. Diese wechselt ihren Wert dann zu 0, 1 oder beliebig (Zeichen: -). In Abbildung 2.2 ist links ein Beispiel für ein KV Diagramm und die Minimierungsterme zu sehen.
Eine Wahrheitstabelle wird rechts auf Abbildung 2.2 durch Nummerierung eines Indexwerts dargestellt. Die Erläuterungen für Minimierungen sind nicht ganz so einfach nachvollziehbar und unübersichtlich dargestellt.

[3] KVD - Karnaugh-Veitch-Diagramm: `https://play.google.com/store/apps/details?id=com.mhsoft.kvd&hl=de`

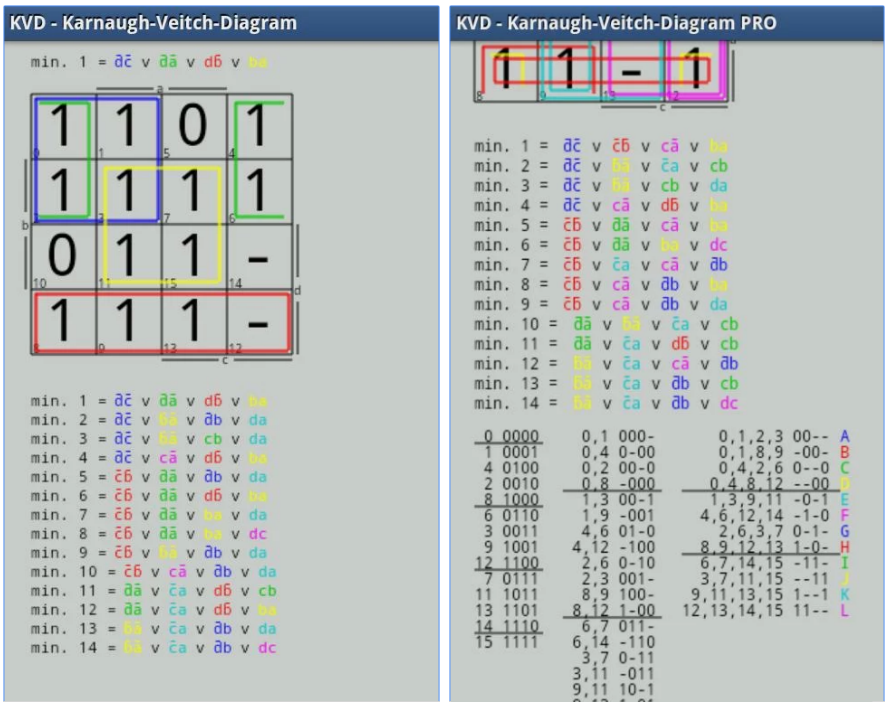


Abbildung 2.2: KVD - Screenshot [GooglePlay (b)]

2.2 Browser

Da es sich in diesem Projekt um eine Webapplikation handelt wurden bei der Recherche auch webbasierte Lösungen miteinbezogen.

2.2.1 Boolesche Ausdrücke vereinfachen

Die Webseite der *Elektroniker Bude*[4] bietet unter dem Menüpunkt *Simulation u. Rechner*[5] zwei Tools für den Umgang mit der Booleschen Algebra. Beide konzentrieren sich auf die Vereinfachung von Booleschen Ausdrücken.
Das erste löst die Vereinfachung über Axiome[6] und das zweite über das KV Diagramm[7]. Von beiden Tools wurde mit einem Screenshot-Ausschnitt auf das Wesentliche reduziert und in Abbildung 2.3 zusammengebracht. In beiden Tools werden im oberen Textabschnitt (wurde beim Screenshot ausgelassen) die Syntax-Regeln beschrieben. Dabei steht * für *UND*, + für *ODER* und ˜ für *NICHT*. Wenn zwei Variablen aufeinander folgen, so wird dazwischen ein *UND* interpretiert.

[4]Elektroniker Bude: `http://www.elektroniker-bu.de/`
[5]Elektroniker Bude - Simulation u. Rechner: `http://www.elektroniker-bu.de/simulation.htm`
[6]Elektroniker Bude: Boolesche Axiome: `http://www.elektroniker-bu.de/boolesche.htm`
[7]Elektroniker Bude: KV Diagramm: `http://www.elektroniker-bu.de/kvdiagramm.htm`

Nach der Eingabe eines Booleschen Ausdrucks und Betätigung des *Berechnen* Buttons wird das neue Ergebnis ausgegeben. Diese Tools führen die Zwischenschritte und das Endergebnis auf und sind somit ein nützliches Mittel um das eigene Ergebnis zu kontrollieren.

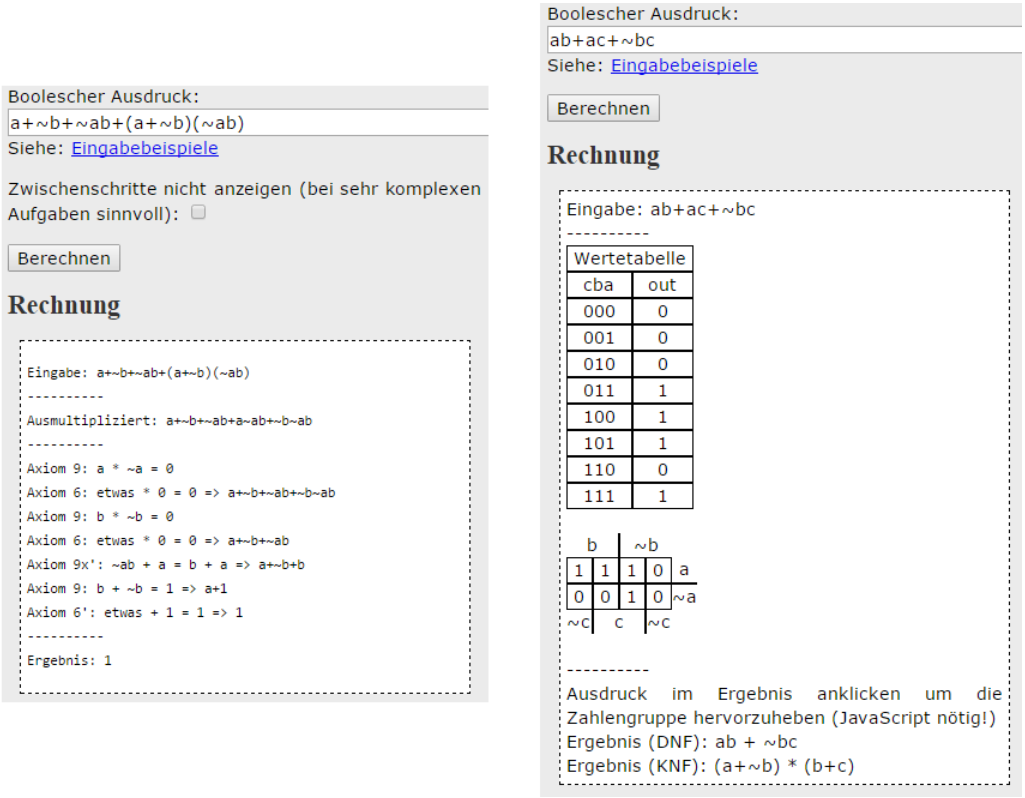


Abbildung 2.3: Elektroniker Bude - Screenshotausschnitte [E-Bude]

2.2.2 Karnaugh-Veitch Map

Die Karnaugh-Veitch Map[8](s. Abbildung 2.4) wurde für die Veranstaltung Technische Informatik[9] an der Universität Marburg als interaktive Demonstration entwickelt. Es sind die Feldgrößen *1x1* bis *8x8* über eine Auswahlbox für das KV Diagramm einstellbar. Durch das Anklicken eines Feldes wird der Wert abwechselnd auf 0 oder 1 gesetzt. Gleichzeitig werden alle Blöcke, die minimiert werden können, farblich gekennzeichnet und die dazugehörige minimierte Form unterhalb des Feldes aufgeführt.
Es kann ebenfalls durch eine Auswahlbox eine *Allow-Don't-Care*-Funktion aktiviert werden. Diese erlaubt neben der Belegung von 0 und 1 auch ein Kreuz zu setzen. Das

[8]Karnaugh-Veitch Map: `http://www.mathematik.uni-marburg.de/~thormae/lectures/ti1/code/karnaughmap/`
[9]Universität Marburg TI: `http://www.uni-marburg.de/fb12/arbeitsgruppen/grafikmultimedia/lehre/ti`

Kreuz ist sowohl mit 0, als auch mit 1 bei der Minimierung kombinierbar. Ist eine zufällige Belegung gewünscht, kann diese durch das Betätigen des *Random example* Buttons durchgeführt werden. Die *Hide*-Funktion blendet den minimierten Ausdruck, der unter dem Diagramm zu finden ist, aus. Im unterem Textabschnitt können die dazu gehörigen JavaScript Dateien heruntergeladen werden.

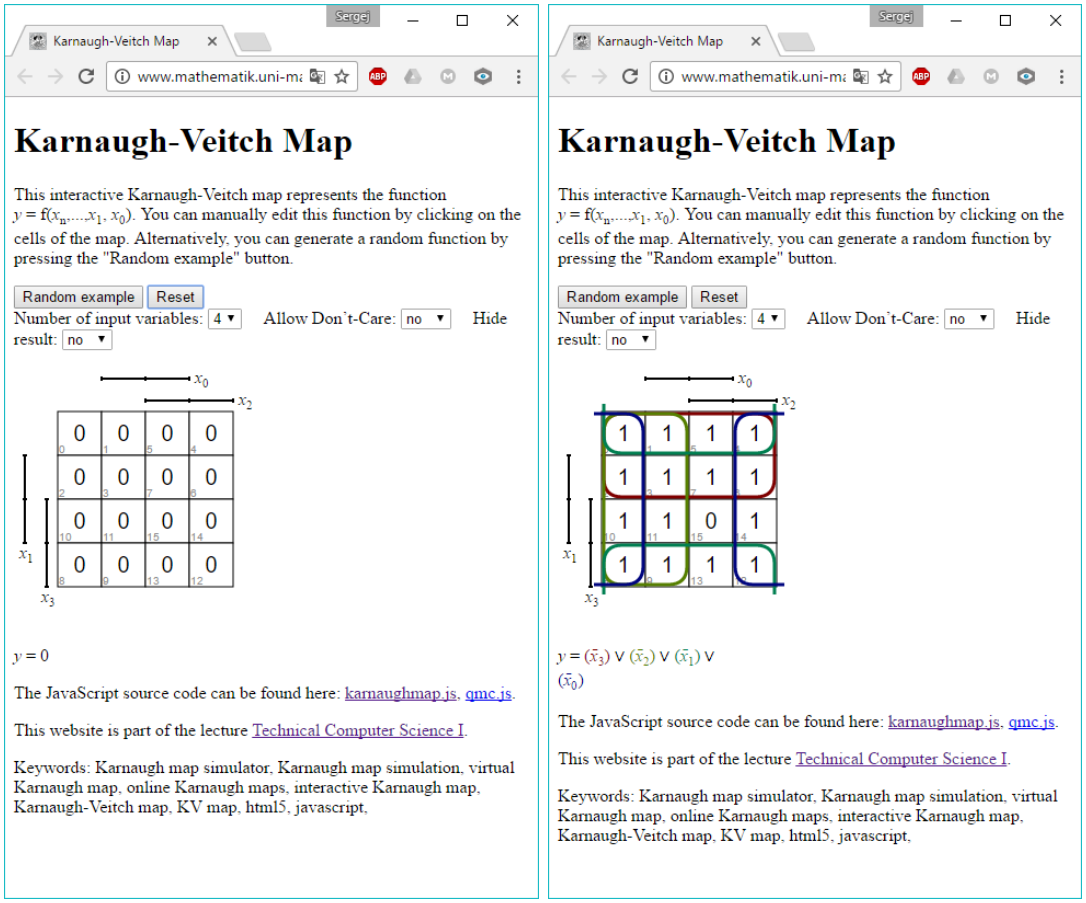


Abbildung 2.4: Universität Marburg - Karnaugh-Veitch Map [Marburg]

3 Vorstellung der vorhandenen Lernplattform

Dieses Kapitel dient zur Vorstellung der bereits vorhandenen Lernplattform (in Unterkapitel 3.1), inklusive enthaltener Lernmodule (Abschnitt 3.1.1, 3.1.2 und 3.2).

3.1 Lernplattform

Die unterschiedlichen Lernmodule werden modular entwickelt. Dadurch steht es offen, diese als einzelne Applikationen zu betrachten oder in eine gemeinsame Applikation zusammenzuführen.
Ziel der Lernplattform[1] ist es eine Sammlung an interaktiver Applikationen zur Verfügung zu stellen, die öffentlich zugänglich sind. Jede Applikation richtet sich genau auf ein bestimmtes Gebiet aus und ermöglicht durch dessen Benutzung eine Vertiefung des Verständnisses. Derzeit werden zwei Themengebiete der Informatik mit jeweils zwei Lernmodulen angeboten. Diese werden im folgenden aufgelistet und kurz beschrieben.

3.1.1 Theoretische Informatik

In diesem Themengebiet gibt es derzeit zwei Lernmodule. Der *Simulator for Finite-State Machines*[2] ermöglicht die Simulation von Endlichen Automaten. Diese können mit vektorgrafikbasierten Linien und Knoten (HTML5-SVG[3]) zusammengesetzt, gespeichert und geladen werden. Bei der Simulation des Vorgangs kann dann ein Eingabewort auf dessen Gültigkeit für diesen Automaten geprüft werden.
Das zweite Lernmodul aus diesem Themengebiet ist der *Simulator for Boolean Algebra*[4], welcher in diesem Projekt erweitert und deshalb separat in Kapitel 3.2 genauer aufgeführt wird.

[1] Lernplattform: `http://lwibs01.gm.fh-koeln.de/blogs/eisemann/learning-modules/`
[2] Simulator for Finite-State Machines: `http://www.gm.fh-koeln.de/~eisemann/eLearning/TI/EA/`
[3] HTML5-SVG: `http://www.w3schools.com/html/html5_svg.asp`
[4] Simulator for Boolean Algebra: `http://www.gm.fh-koeln.de/~eisemann/eLearning/TI/Boole/`

3.1.2 Computergrafik und Animation

Computergrafik und Animation ist ein Schwerpunkt der Medieninformatik an der TH Köln. Im ersten Modul werden die Grundlagen von *Texture Mapping*[5] näher gebracht. Das *CG Learning Modules*[6] beinhaltet eine große Sammlung von Modulen im Bereich der Computergrafik, deren einzelne Erläuterungen den Rahmen sprengen würden.

3.2 Lernmodul für Boolesche Algebra

In diesem Abschnitt wird das Lernmodul, was (wie in Kapitel 1.1 bereits erwähnt) im Praxisprojekt (s. Datenträger) umgesetzt wurde, vorgestellt.
Der *Simulator for Boolean Algebra* ermöglicht die Eingabe von Booleschen Ausdrücken. Operatoren können mit Hilfe von Buttons eingesetzt werden und um die Möglichkeit zu erhalten, Zwischenschritte anwenden zu können, wurde ein Button für das Gruppieren von Teilausdrücken zur Verfügung gestellt.
Basierend auf dem eingegebenem Booleschen Ausdruck wird eine Wahrheitstabelle generiert, in der dann Ergebnisse vom Benutzer eingetragen und überprüft werden können. In Abbildung 3.1 wurde der Ausdruck A∧B∨C getestet, wobei A∧B zu G1 gruppiert wurden.

Abbildung 3.1: Screenshot von Simulator for Boolean Algebra [eLearning Boolean Alg]

[5] Texture Mapping: http://www.gm.fh-koeln.de/~eisemann/eLearning/CGA/TextureMapping/
[6] CG Learning Modules: http://cg.web.th-koeln.de/cg-learning/index.php

4 Konzept

Das Gesamtkonzept ist eine Lernplattform (s. Kapitel 3), die ein interaktives Lernen von Themengebiete in der Informatik ermöglicht.
Das Lernmodul (s. Kapitel 3.2) vorgestellt wurde, wird in diesem Projekt erweitert. Um die Funktionsweise der möglichen Erweiterungen klarzustellen und keine Funktion zu übersehen, werden die dazugehörigen theoretischen Grundlagen in den nachfolgenden Abschnitten kurz erläutert. Anschließend wird auf das Konzept dieser Erweiterungen eingegangen und der Umsetzungsumfang eingegrenzt.

4.1 Theorie

Um Begriffe von Beginn an festzulegen und Ziele besser definieren zu können, werden in nachfolgenden Abschnitten die wichtigsten Grundlagen zu *Boolescher Ausdruck*, *Wahrheitstabelle*, *KV Diagramm*, *Minimierung im KV Diagramm* und *Schaltsysteme* kurz in Bezug auf die Erweiterungsziele erläutert.
Die Theorie wird in möglichst einfachen Worten erklärt und orientiert sich nach den Büchern *Grundlagen der Technischen Informatik* [Hoffmann (2014)] und *Theoretische Informatik* [Hoffmann (2015)] von Dirk. W. Hoffmann.
Die Beispielgrafiken wurden mit dem Online Tool draw.io[1] erstellt.

4.1.1 Boolescher Ausdruck

Ein Boolescher Ausdruck ist ein mathematischer Ausdruck in der Booleschen Algebra. Dabei werden statt den gängigen Rechenoperatoren, wie zum Beispiel Addition und Subtraktion, logische Operatoren verwendet. Die für dieses Projekt benötigten logischen Grundoperatoren sind $\wedge$ *(UND)*, $\vee$ *(ODER)* und $\neg$ *(NICHT)*. Weitere Operatoren sind lediglich spezielle Formen oder Kombinationen dieser Grundoperatoren.

4.1.2 Wahrheitstabelle

Die Wahrheitstabelle wird dafür verwendet, um für jede mögliche Zuordnung einer Variable ein Ergebnis festzuhalten. Dabei bekommt jede enthaltene Variable eine eigene

[1] draw.io: `http://draw.io/`

Spalte in der Tabelle. Die letzte Spalte in der Tabelle ist für das jeweilige Ergebnis des Booleschen Ausdrucks reserviert.
Nachdem alle Spaltennamen festgelegt wurden, folgt die Belegung von binären Ziffern in jeder Spalte, ausgenommen der Ergebnisspalte. Bei dieser Belegung wird jede mögliche Kombination aller Variablen mit den Werten 0 und 1 aufgeführt. Im Anschluss wird jede Kombination in den Booleschen Ausdruck eingesetzt und das Ergebnis in der Ergebnisspalte festgehalten. In Abbildung 4.1 ist ein Beispiel mit dem Booleschen Ausdruck F=A∧B dargestellt.
Bei größeren Booleschen Ausdrücken können Teilausdrücke gruppiert werden und dessen Ergebnis ebenfalls in der Tabelle zwischen den Variablen und dem Endergebnis festgehalten werden.

$F = A \wedge B$

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

Abbildung 4.1: Wahrheitstabelle für F=A∧B

4.1.3 KV Diagramm

Ein KV Diagramm ist eine andere Darstellung für einen Booleschen Ausdruck.
Boolesche Ausdrücke und Schaltungen können, wenn sie nicht optimal minimiert sind, mit verschiedenen Techniken verkleinert werden. Eine effiziente Methode ist das Konstruieren eines KV Diagramms, mit dessen Hilfe dann der dazugehörige Ausdruck minimiert werden kann.
Um ein KV Diagramm zu erstellen, ist zum gewünschten Ausdruck zunächst eine Wahrheitstabelle (s. Kapitel 4.1.2) aufzustellen. Sind die Ergebnisse für jede mögliche Belegung ermittelt, kann das KV Diagramm erzeugt werden. Dazu werden alle Variablen im Ausdruck (ausgenommen vom Ergebnis und Zwischenschritte/Gruppen) zu einer Menge zusammengefasst. Damit KV Diagramme besser miteinander verglichen werden können, ohne umzudenken oder sie umzustellen zu müssen, werden die Variablen der Menge in die Reihenfolge gebracht, in der sie im Booleschen Ausdruck stehen. Jedoch spielt die Reihenfolge für die Funktionalität keine Rolle, solange das KV Diagramm korrekt anhand der Wahrheitstabelle aufgebaut wurde.
Ein KV Diagramm ist optisch eine große Tabelle mit 2^n Zellen (n steht für die Anzahl der Variablen). Jede Zelle steht für eine andere Variablenkombination und enthält als Wert das Ergebnis dieser Kombination. Die Variablen, die der Zelle zugewiesen sind,

stehen dann je nach Wunsch oberhalb, unterhalb, links oder rechts vom Diagramm. In diesem Projekt stehen die Variablen links und oberhalb der Box.
Um das KV Diagramm zu erstellen ist folgende Vorgehensweise einzuhalten: Mit dem Erweitern um eine Variable wird das bereits vorhandene Feld um Zellen erweitert. Zuallererst werden zwei Zellen in einer Zeile erstellt.
Die erste Variable aus der Menge wird dann in den Tabellenkopf über der ersten Zelle des KV Diagramms geschrieben. Die zweite Zelle erhält die Variable in negierter Form. Für die zweite Variable aus der Menge werden erneut zwei Zellen erstellt, die genau unter den zuvor erstellten Zellen liegen. Die zweite Variable wird diesmal vertikal links neben die Tabelle des KV Diagramms geschrieben (Beispiel für zwei Variablen ist in Abbildung 4.2 zu sehen).
Bei jeder weiteren Variable wird das KV Diagramm abwechselnd horizontal und vertikal gespiegelt (Variablenzuweisung bleibt bestehen). Nach der Spiegelung wird dann die neue Variable in normaler Form den alten, in negierter Form den neuen Zellen zugewiesen und je nach Spiegelung dann oberhalb und links von der Tabelle aufgeschrieben (Beispiel für drei Variablen ist in Abbildung 4.3). Ist die Form des KV Diagramms erstellt, folgt die Belegung der Werte. Dafür wird die Wahrheitstabelle benötigt. Ist bei einer Zelle die Variable negiert, so entspricht das in der Wahrheitstabelle der Spalte, wo eine 0 vorkommt und der Spaltenkopf die Variable ist.

$F = A \wedge B$

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

	A	¬A
B	1	0
¬B	0	0

Abbildung 4.2: KV Diagramm für F=A∧B

$F = A \wedge B \vee C$

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

	C	C	¬C	¬C
	A	¬A	¬A	A
B	1	1	0	1
¬B	1	1	0	0

Abbildung 4.3: KV Diagramm für F=A∧B∨C

4.1.4 Minimierung im KV Diagramm

Für die Minimierung eines Schaltsystems können unterschiedliche Techniken angewendet werden. Eine dieser Techniken ist der Einsatz eines KV Diagramms. Mit Hilfe dieses Diagramms kann eine Disjunktive (DNF) und eine Konjunktive Normalform (KNF) erzeugt werden, die dann jeweils eine minimierte Version des ursprünglichen Ausdrucks repräsentieren.
Bei der Minimierung werden möglichst große zusammenstehende Blöcke der Größenordnung 2^n aus dem KV Diagramm rausgesucht. Je nachdem, ob eine DNF oder KNF gewünscht ist, werden nur die Zellen mit dem Wert 1 oder 0 betrachtet.
In Abbildung 4.4 wird die Minimierung für den Ausdruck F:=A∧(B∨¬B)∧C∨D vorgestellt. Dabei wird das DNF und KNF gebildet. Beim DNF werden die Zellen mit dem Wert 1 betrachtet. Ist in einem Block sowohl die positive als auch die negative Variante einer Variable vorhanden, so fällt diese aus dem Block raus. Beim KNF werden die 0en und die Variablen negiert betrachtet. Anschließend werden die Blöcke beim DNF mit einer Disjunktion (Oder) und beim KNF mit einer Konjunktion (Und) verbunden.

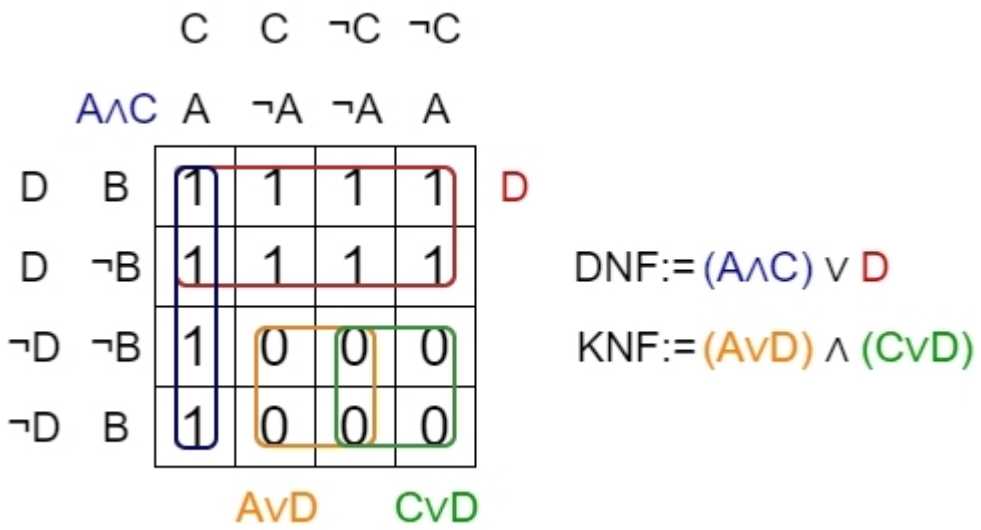


Abbildung 4.4: F=A∧(B∨¬B)∧C∨D Minimierung

4.1.5 Schaltsysteme

Boolesche Ausdrücke können in Schaltungen übersetzt werden. Eine Schaltung besteht aus logischen Gattern (auch Logikgatter genannt). Diese Gatter sind elektronische Bauelemente, die zum Beispiel auf einer Chipplatine zur Signalübertragung verbaut werden. Logikgatter verarbeiten digitale Eingabesignale zu einem vom jeweiligen Gatter abhängigem digitalen Ausgabesignal. Die Variablen eines booleschen Ausdrucks werden zur Signalquelle und der Operator zum Logikgatter einer Schaltung. In Abbildung 4.5 sind einige Hauptgatter und der dazugehörige boolesche Ausdruck aufgeführt.

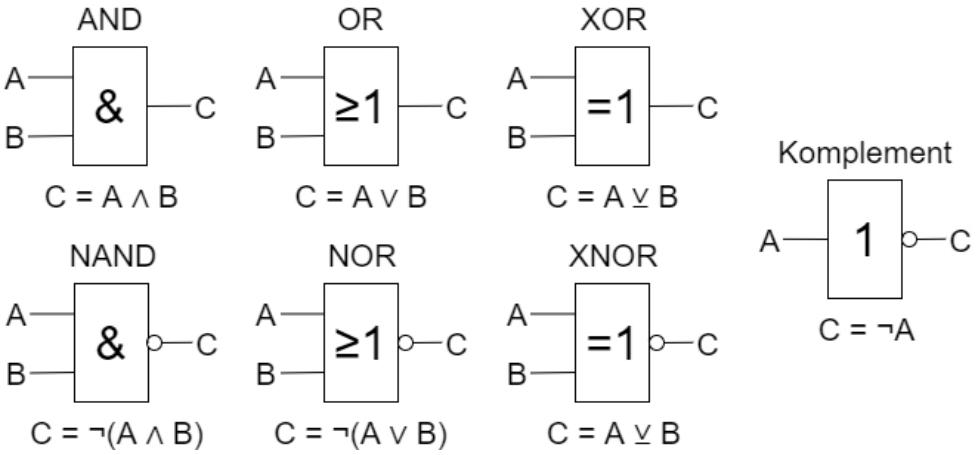


Abbildung 4.5: Logikgatter

Hat der boolesche Ausdruck mehr als nur einen Operator, können dann Gatter zu einer gemeinsamen Schaltung zusammengeführt werden. Ein Beispiel ist in Abbildung 4.6 zu sehen.

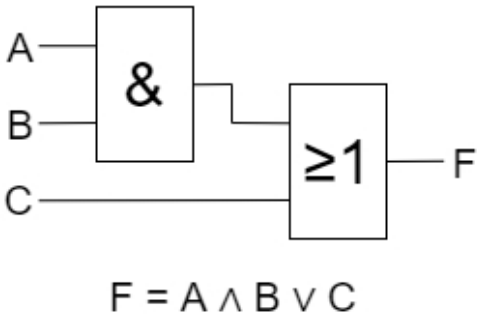


Abbildung 4.6: Schaltung von F=A∧B∨C

4.2 Designüberlegungen

Dieses Kapitel beschäftigt sich sowohl mit dem technischen, als auch optischen Design der Applikation. Zuerst werden in Abschnitt 4.2.1 die zuvor angefertigten Mockups beschrieben. Danach wird auf den Systementwurf (s. Kapitel 4.2.3), eingegangen. Passend zu dem Entwurf werden potenzielle Komponenten aufgeführt. Anschließend folgen in Kapitel 4.2.4 Überlegungen zum Design.

4.2.1 Mockups

In diesem Unterkapitel werden Mockups vorgestellt, die bereits im Praxisprojekt angefertigt, jedoch nicht dokumentiert, wurden. Diese wurden entworfen um eine grobe Orientierung darüber zu erhalten, wie die Webapplikation im Endstadium aussehen könnte. Dabei wurde sich aus dem Bereich der Booleschen Algebra auf Booleschen Ausdruck, Wahrheitstabelle, KV Diagramm und Schaltsysteme beschränkt.
Um alle Komponenten auf einem Blick zu erhalten wurden die Mockups in der Desktop-Ansicht gestaltet. Für die Gestaltung wurde die Applikation Balsamiq[2] verwendet. Im Verlauf der Mockupvorstellung wird nur im ersten (s. Abbildung 4.7) und im letzten Mockup (s. Abbildung 4.10) das gesamte Layout angezeigt.

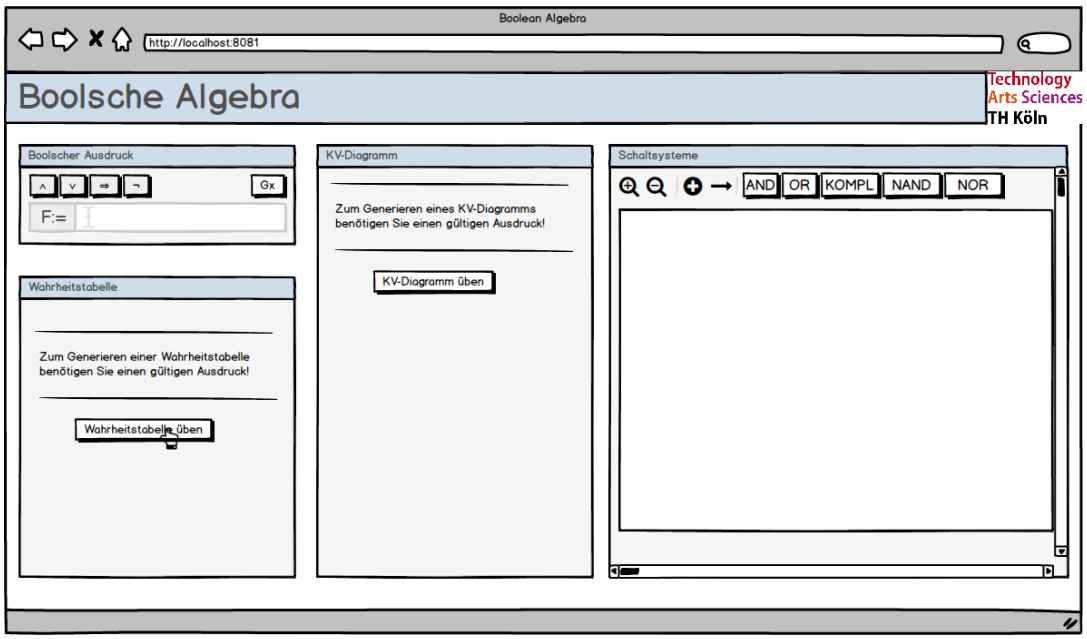


Abbildung 4.7: Mockup - Startansicht

[2]Balsamiq: `https://balsamiq.com/`

In Abbildung 4.7 wird die Startansicht dargestellt, die erscheint, wenn die Applikation geöffnet wird. Diese ist in vier Boxen unterteilt. Jede Box repräsentiert ein Thema der oben genannten Themen der Booleschen Algebra. Die Reihenfolge der Boxen ist durch die Vorgehensweise bestimmt, die genutzt werden würde, um von einem Booleschen Ausdruck über das KV Diagramm eine minimierte Schaltung zu erhalten. Bei jedem nächsten Schritt (nächste Box) steht ein Hinweis, was zu erfüllen ist, damit diese Funktion abgerufen werden kann. Zum Beispiel kann keine Wahrheitstabelle ohne einen gültigen Booleschen Ausdruck erstellt werden.

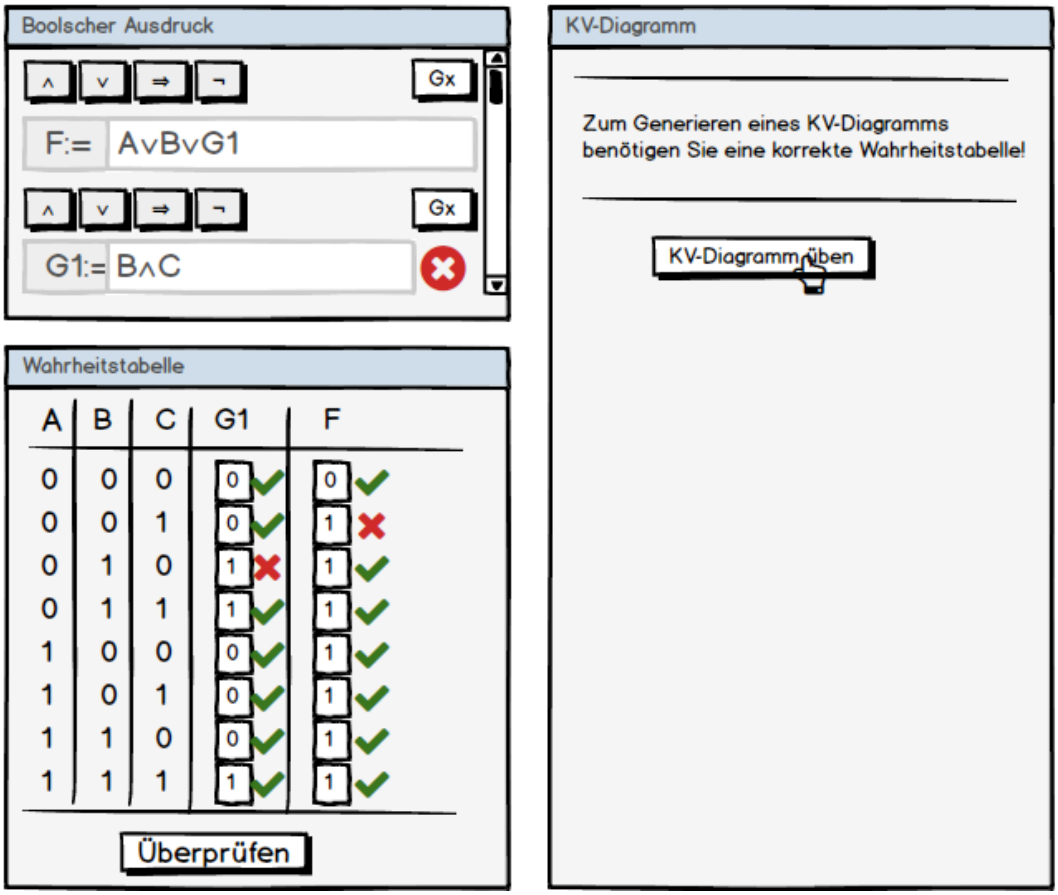


Abbildung 4.8: Mockup - Wahrheitstabelle

Abbildung 4.8 zeigt, wie es aussehen könnte, wenn aus dem eingegebenen Booleschen Ausdruck eine Wahrheitstabelle generiert wird. Dabei ist zu beobachten, dass ein Teil des Ausdrucks gruppiert wurde, um zu Veranschaulichen, wie mit Gruppen hantiert werden könnte. Jede Gruppe bekommt ein eigenes Eingabefeld und Operator-Buttons. Mit dem roten Kreuz rechts neben dem jeweiligen Textfeld kann die Gruppe gelöscht werden.
Bei der Wahrheitstabelle erhalten Gruppen und das Ergebnis eine eigene Spalte und ein

Eingabefeld für das Ergebnis. Die Variablen werden automatisch mit 0 und 1 belegt. Nach Betätigen des *Überprüfen* Buttons werden die Eingabefelder der Gruppen und von F auf dessen Korrektheit geprüft. Ist die Wahrheitstabelle korrekt, kann durch Betätigen des *KV Diagramm üben* Button das dazu passende KV Diagramm generiert werden.
Aus der in Abbildung 4.8 erstellten Wahrheitstabelle wird in Abbildung 4.9 ein KV Diagramm erstellt. Dort stehen Farben für das Erstellen von Blöcken innerhalb des KV Diagramms zur Verfügung. Die farblichen Blöcke, die der Benutzer selber zusammenstellen kann, geben diesem Hilfestellung bei der Erstellung der KNF und DNF. Mit diesen Normalformen kann dann der minimierte Ausdruck bestimmt werden. Für die Eingabe sind jeweils ein Eingabefeld und Operator-Buttons platziert.

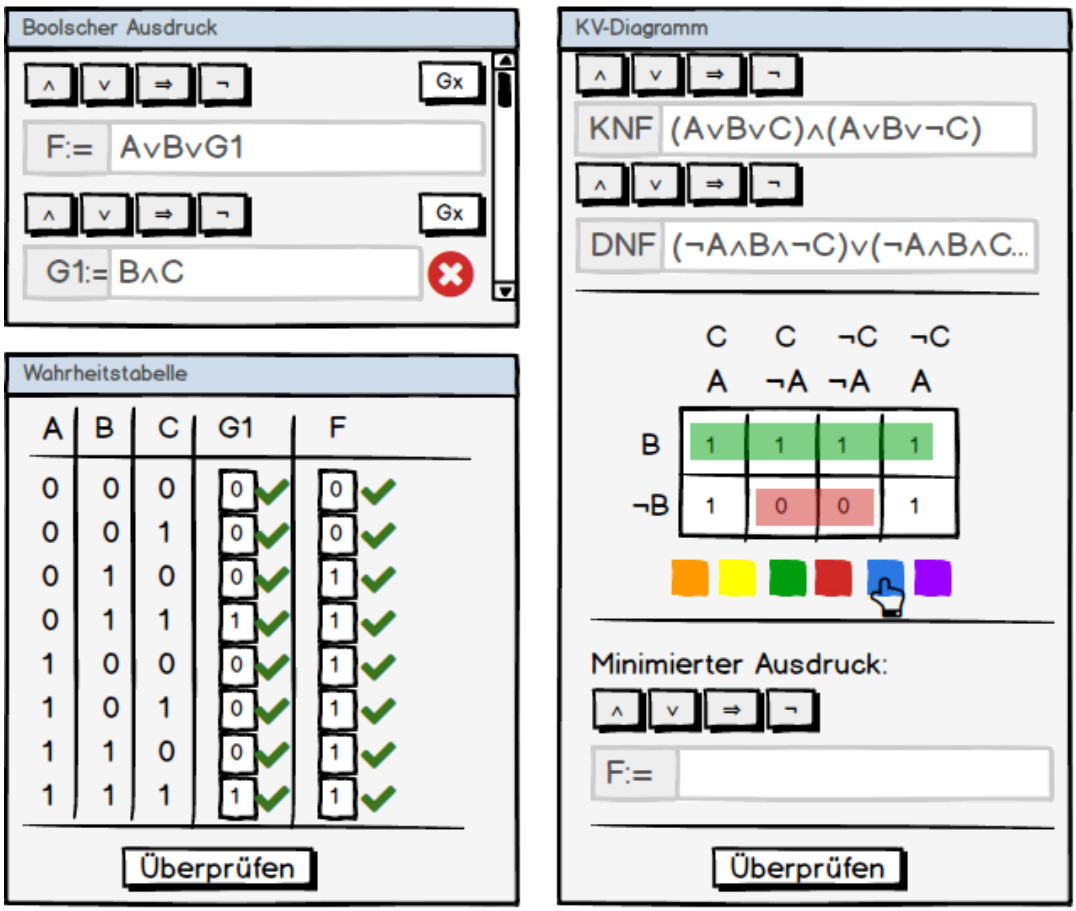


Abbildung 4.9: Mockup - KV Diagramm

Auf Basis des minimierten Terms kann der Benutzer nun eine logische Schaltung (Kapitel 4.6) erstellen. Für die Konstruktion stehen ihm Buttons zur Verfügung, mit denen die einzelnen Bauteile platziert werden können. Im Anschluss können dann diese durch

Linien verbunden werden. Wie dies aussehen könnte ist in Abbildung 4.10 zu betrachten.

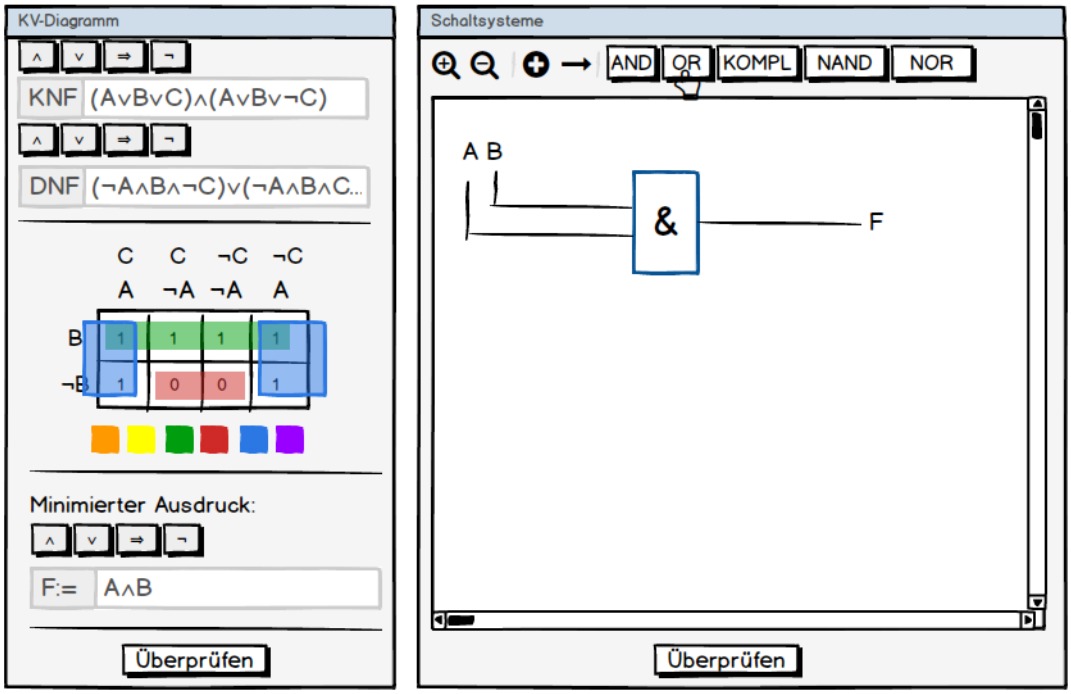


Abbildung 4.10: Mockup - Schaltsysteme

4.2.2 Anpassung der Mockups

Die in Abschnitt 4.2.1 aufgeführten Mockups wurden zu dem Zeitpunkt entwickelt, als es lediglich eine genauere Planung für das Eingabefeld des Booleschen Ausdrucks und der Wahrheitstabelle gab. Das Skizzieren des KV Diagramms und der Schaltsysteme erfolgte rein aus Orientierungsgründen für das Endprodukt. Da sich dieses Projekt mit der Umsetzung weiterer Funktionen beschäftigt sind Anpassungen angebracht. Nach Absprache für die Umsetzung des KV Diagramms wurde beschlossen, dass die Möglichkeit bestehen sollte bei farblichen Ausdrücken den dazu passenden Ausdruck einzugeben. Außerdem wurde anstelle vom Eingabefeld für den minimierten Ausdruck die Felder vom DNF und KNF platziert. Dieses wird vorerst in Abbildung 4.11 festgehalten.

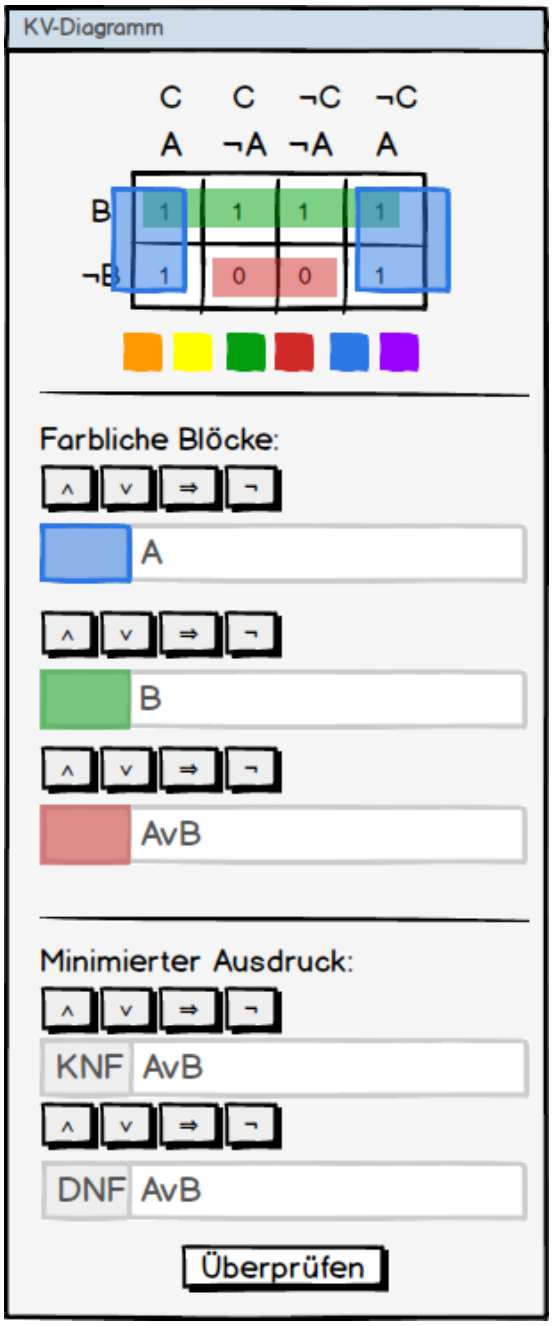


Abbildung 4.11: Mockup - KV Diagramm (bearbeitet)

4.2.3 Systementwurf

Wie bereits in Kapitel 4.2 erwähnt, wird auf den Systementwurf im Praxisprojekt eingegangen. Dort wurden die Komponenten AngularJS Direktiven, AngularJS Controller und AngularJS App erläutert. Um diese Webapplikation nun zu erweitern sind weder weitere AngularJS Controller, noch eine AngularJS App notwendig. Es Bedarf lediglich

der AngularJS Direktiven, da die Erweiterungen Teile der gesamten Applikation sind.

Potenzielle Komponenten
Folgende Komponenten könnten für die Entwicklung in Frage kommen:

- Gesamtbereich des KV Diagramms als Direktive. In dieser Direktive soll der Zusammenhang zwischen allen Unterdirektiven behalten werden.
- Eine Direktive, die die Direktive *boolExpr* aus dem Praxisprojekt verwendet, um eine individuelle Erweiterung bezüglich des KV Diagramms in die Texteingabemaske zu übermitteln.
- Alles zu Schaltsystemen unterliegt ebenfalls einer separaten Direktive.

4.2.4 Optisches Design

Beim optischen Design sollte auf Einheit mit der vorhandenen Applikation geachtet werden. Bereits vorhandene Designstrukturen und Regeln sollten möglichst wiederverwendet werden.
Das KV Diagramm sollte in der Horizontalen immer zentriert ausgerichtet sein, damit die Fläche gut genutzt wird. So wird bei jeder Größe die Fläche rechts und links neben dem KV Diagramm gleich belegt. Die Farben für die Gruppierung von Blöcken sollten deutlich unterschieden werden können.

4.3 Überlegungen zur Datenstruktur des KV Diagramms

Mit der Datenstruktur soll ein KV Diagramm (wie in Kapitel 4.1.3 beschrieben) dargestellt werden können. Es wird also ein Netz mit nummerierten Zellen benötigt, dass jede Zelle mit dessen horizontalen und vertikalen Nachbarn verbindet. Außerdem ist es praktisch, wenn die letzte Zelle einer Zeile mit der ersten derselben Zeile und die Zellen der ersten Zeile mit denen der gleichen Spalte und letzten Zeile verbunden ist. Dies könnte unter Umständen Rechenzeit sparen, weil Zellen nicht mehrfach indexiert werden müssen. Es wird stattdessen ein schneller Zugriff durch Referenzen gegeben. Um dies bildlich zu verdeutlichen ist in Abbildung 4.12 ein Beispiel dargestellt. Des Weiteren soll das Netz durch Spiegelung aufgebaut ist. In Abbildung 4.12 wurde bei jeder Spiegelung eine neue Farbe angewendet. Dort wurde das KV Diagramm mit den Variablen A, B, C, D und E aufgebaut. Die neu entstandenen Zellen und Buchstaben sind dabei mit der Spiegelungsfarbe versehen.

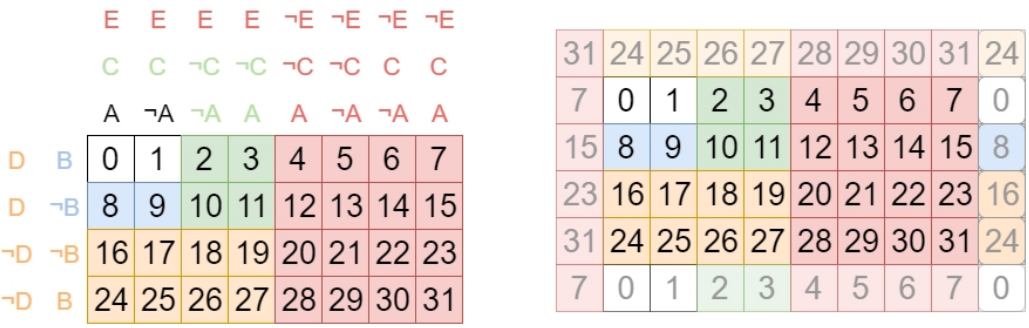


Abbildung 4.12: KV Diagramm Netz

4.4 Suche von größten Blöcken

Um das Ergebnis zu überprüfen und die vom Benutzer erstellten farblichen Blöcke korrekt darstellen zu können, ist eine Suche nach der optimalen Lösung seitens der Applikation notwendig. Um die optimale Lösung zu finden, müssen die Zellen nach den größten Blöcken gruppiert werden.
Die Suche muss durch Spiegelung erzeugt werden. In diesem Fall bedeutet Spiegelung, dass wenn ein Block n Zellen hat, dann auch die nächsten n Nachbarn mit dem gleichen Wert, geprüft werden und bei Validität diesen Block erweitern. Ist die Rede von *inneren Zellen*, so sind die gemeint, die sich innerhalb des KV Diagramms befinden. Die Zellen, die außerhalb des KV Diagramms weitergeführt werden (wie in Abbildung 4.12 zu sehen) werden hier als *äußere Zellen* bezeichnet.
In den nachfolgenden Unterkapiteln sind drei Denkansätze für mögliche Algorithmen aufgeführt.

4.4.1 Rekursive Suche

Die erste Überlegung ist, das KV Diagramm rekursiv in alle vier Richtungen suchen zu lassen. Die aus dem vorherigen Schritt werden dann jeweils gespiegelt. Gemeinsame Blöcke werden anschließend zusammengefasst. Diese Idee bringt jedoch ein Übel mit sich: die Suche je nach Zellanzahl exponentiell ansteigen und Zellen werden unter Umständen mehrfach besucht. Dadurch werden ähnliche Blöcke erzeugt, die dann bei der Zusammenführung wieder ausgeschlossen werden. Die Fehlersuche während der Umsetzung könnte sich ebenfalls als problematisch erweisen, da bei Rekursion einzelne Schritte schwierig nachvollziehbar sind. Würde der Algorithmus darauf reduziert werden in zwei, statt in vier Richtungen zu suchen, würden die Probleme lediglich halbiert werden und weiterhin bestehen.

4.4.2 Iterative Bruteforce-Suche

Eine bessere Idee wäre eine iterative Bruteforce-Suche. Bei dieser Methode werden von jeder Zelle aus nach unten und rechts die möglichen Schritte ausprobiert und alle Blöcke der Größe 2^n zusammengestellt. Anschließend werden aus diesen Blöcken die größten rausgesucht, bei denen Zellen vorhanden sind, die in den vorher gewählten Blöcken nicht vorhanden sind.
Bei dieser Methode könnten jedoch auf Grund von mehreren verschachtelten Schleifen und mehrfacher Indexierung bei größeren KV Diagrammen Performanzeinbrüche entstehen.

4.4.3 Iterative Suche durch Referenzen

Um einen exponentiellen Wachstum an Blöcken und mehrfach verschachtelte Schleifen zu verhindern, könnte stattdessen eine iterative Herangehensweise in einem verketteten Zellnetz durch Referenzen Abhilfe verschaffen. Hierbei müsste darauf geachtet werden, dass jede Blockvariante nur einmal erzeugt wird, wodurch der Vergleich und das Zusammenführen vorhandener Blöcke wegfällt und ebenfalls an Rechenzeit gespart wird. Der Algorithmus wurde darauf eingeschränkt, dass nicht direkt alle möglichen Blöcke erzeugt werden. Es werden Schrittweise die Zellen als Startpunkt gewählt, die noch nicht besucht wurden bzw. in einem Block enthalten sind. Ein Block wird so lange gespiegelt, bis die maximale Größe erreicht ist. Bei der Spiegelung wird priorisiert gehandelt. Die Richtungen verlaufen sich nach rechts, unten, links und oben. Da es aus praktischen Gründen bevorzugt wird, dass Spiegelungen über innere Zellen des KV Diagramms gehen, werden Richtungen bevorzugt, die nicht über den Rand des KV Diagramms verlaufen. Erst wenn dies nicht mehr möglich ist, werden alle Varianten gewählt, die sich mit äußeren Zellen verbinden. Zusammenfassend gesagt, wird nacheinander jede Richtung nach inneren Zellen geprüft. Sind Spiegelungen in Richtung der inneren Zellen mehr möglich, erfolgt das gleiche bei allen äußeren Zellen. Die einzige Voraussetzung dieses Algorithmus ist, dass die Zellen beim Aufbau des KV Diagramms einmalig miteinander verkettet werden müssen (s. Abbildung 4.12 rechts).
In Abbildung 4.13 ist die Suche für den ersten Block (in grün) vollständig und für den zweiten Block (blau) nur der erste Schritt aufgeführt.
In dieser Abbildung ist zu beobachten, dass die Blöcke sich zuerst nach rechts erweitern und sich, falls keine innere Spiegelung mehr möglich ist, nach außen spiegeln. Das Endergebnis kann der Abbildung 4.14 entnommen werden.

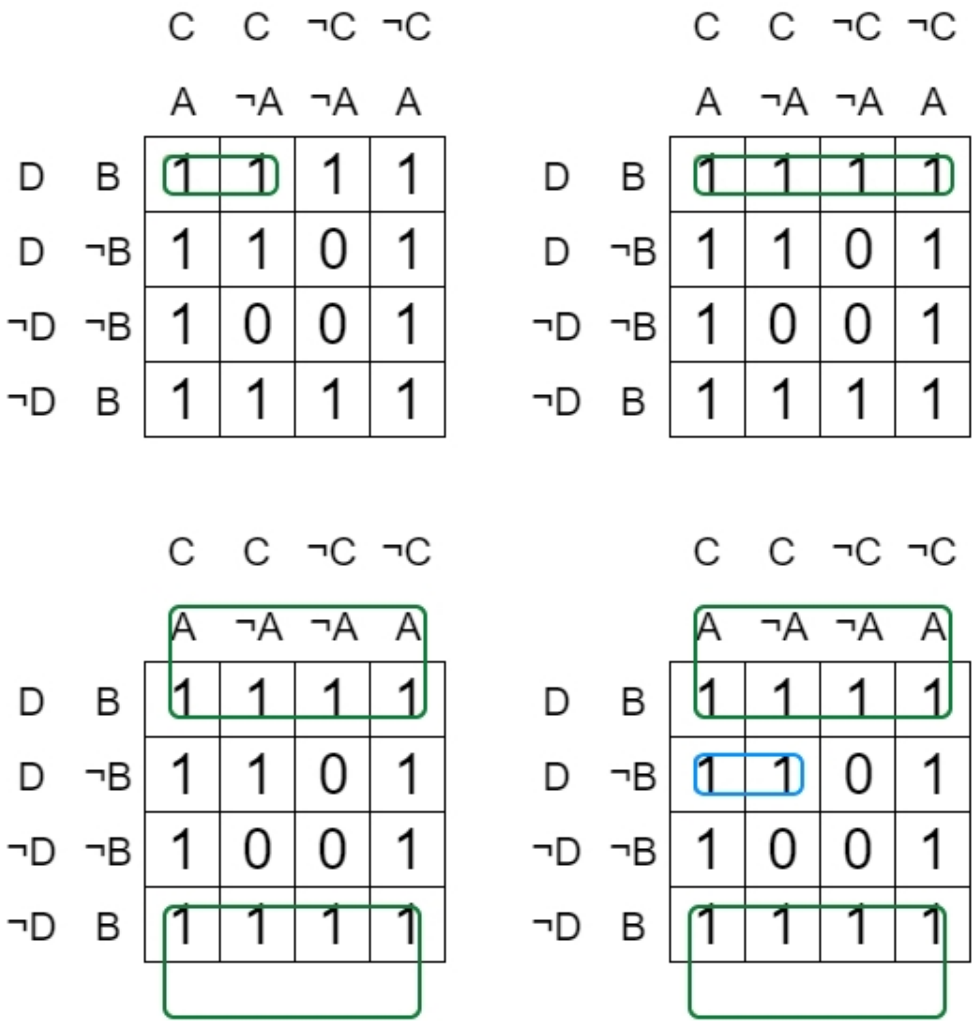


Abbildung 4.13: Iterativer Algorithmus für F=A∨B∨C∧D

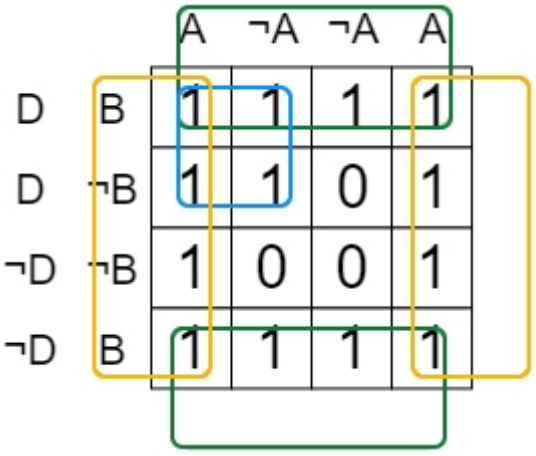


Abbildung 4.14: Lösung für F=A∨B∨C∧D

Anwendungsbeispiel für den Algorithmus

Dieser Vorgang wird nun für den Anwendungsfall in Kapitel 4.1.3 geprüft:
In diesem Vorgang wird die DNF gebildet, wodurch die Zellen mit einer 1 als Wert in Frage kommen. Der Ablauf ist in Abbildung 4.15 durch Kennzeichnung der einzelnen Schritte mit *(a)* bis *(e)* dargestellt.
Beim Start des Algorithmus wird jede Zelle besucht, die sich noch in keinem Block befindet. Da dies der erste Schritt ist, wird die erste Zelle genommen. Bei der Initialisierung (Schritt *a*) wird ein blauer Block in die erste Zelle gelegt. Von nun an wird die Spiegelung für diesen blauen Block in jede Richtung ausprobiert. Da rechts angefangen wird und eine Spiegelung möglich ist (Wert ist der gleiche wie in der Startzelle) wird direkt gespiegelt. Dabei wird nur eine Nachbarzelle geprüft, weil sich im aktuellen Block nur eine Zelle befindet. Dieser Vorgang ist in Schritt *b* zu beobachten. In Schritt *c* wurde der blaue Block nun nach unten gespiegelt, da nach rechts hin keine Spiegelung möglich ist. Eine Spiegelung nach außen ist ebenfalls nicht möglich. Somit ist dieser Block komplett und es wird der nächste erstellt. Der nächste Block wird in der Zelle erzeugt, die sich noch in keinem Block befindet und den Wert 1 hat. Dies ist in Schritt *d* zu entnehmen. Da keine Spiegelung nach innen möglich ist, jedoch eine nach außen, wird diese durchgeführt. Dadurch wird die Zelle mit der ersten Zelle verbunden. Von nun an sind keine weiteren Spiegelungen mehr möglich. Die letzte mögliche Zelle wurde ebenfalls besucht.

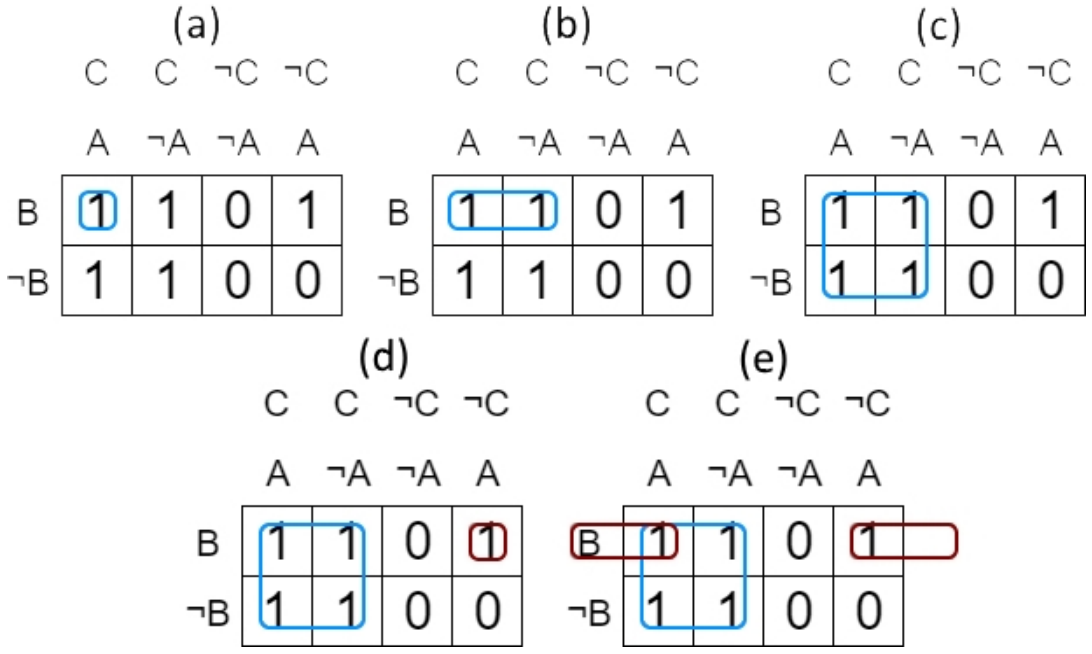


Abbildung 4.15: F=A∧B∨C Algorithmusvorschau

4.5 Umsetzungsentscheidung

Da die Programmierung der Datenstruktur und Algorithmus des KV Diagramms, dessen Integration an die Benutzeroberfläche, das Testen auf unterschiedlichen Geräten und Dokumentation der Arbeit bereits einen hohen Arbeitsaufwand darstellen, wurde sich dagegen entschieden die Schaltsysteme mit einzubringen. Der voraussichtliche zusätzliche Arbeitsaufwand für die Schaltsysteme, die mit vielen grafischen Elementen verbunden sind, würde den Rahmen der Bachelorarbeit sprengen und durch Zeitdruck das Risiko für Fehler erhöhen. Die optimale Vorgehensweise für Schaltsysteme wäre eine Einarbeitung in SVG-Elemente, ähnlich wie im *Simulator for Finite-State Machines*(s. Kapitel 3.1.1). Dieser verwendet das Framework *d3*[3]. Anschließend käme noch eine Validierung der Schaltung dazu.
Folglich wird dieses Projekt auf die Umsetzung des KV Diagramms beschränkt.

[3]d3: `https://d3js.org/`

5 Anforderungsanalyse

Die Anforderungsanalyse dient zur Ermittlung konkreter Ziele, Aufgaben, Funktionen und Rahmenbedingungen für die Entwicklung der Applikation. Diese Anforderungen sind als Richtlinien zu betrachten und können sich im Verlauf der Entwicklung ändern.

5.1 Zielbestimmungen und Zielgruppen

In diesem Abschnitt werden die Produktperspektive, Einsatzkontext und Zielgruppen für die Applikation vorgestellt.

5.1.1 Produktperspektive

Da diese Applikation ein Bestandteil der in Kapitel 3 vorgestellten Lernplattform ist, muss diese ein interaktives Lernen fördern. Dafür muss das gezielte Thema (die Boolesche Algebra) dem Benutzer durch Interaktion näher gebracht werden. Dies bedeutet, dass Herangehensweisen, welche sonst von Hand angewendet werden, in der Anwendung simuliert werden und auf Anfragen seitens Nutzer reagieren. Darunter fallen das Befüllen einer Wahrheitstabelle und Minimierung eines KV Diagramms, anhand von selbst erstellten Blöcken.

5.1.2 Einsatzkontext

Die Applikation soll im Browser sowohl auf mobilen Endgeräten, als auch auf einem Desktop Computer und Laptop laufen. Dadurch ist keine Installation seitens Nutzer und Portierung seitens Entwickler notwendig.

5.1.3 Zielgruppen

Grundsätzlich könnte die Applikation für alle interessant sein, die sich für Boolesche Algebra interessieren oder ihre Ergebnisse vom Papier in der Applikation validieren möchten. Jedoch ist der Fokus hauptsächlich auf Informatikstudenten gelegt, da Zwischenschritte nicht erklärt werden und kein Einführungstext vorhanden ist. Die Applikation setzt also voraus, dass der Nutzer versteht, was ein Boolescher Ausdruck und

dessen Operanden und Operatoren sind. Außerdem ist das Wissen über den Umgang mit einer Wahrheitstabelle und dem KV Diagramm nützlich.

5.2 Funktionale Anforderungen

Die funktionalen Anforderungen sind die von der Applikation zu erbringenden Funktionen. Diese werden im folgenden zusammengefasst und können von den früher erstellten Mockups teilweise abweichen:

- Der Benutzer soll nach Eingabe eines gültigen Booleschen Ausdrucks das dazu gehörende KV Diagramm generieren lassen
- Für die Minimierung des Booleschen Ausdrucks sollen farbliche Blöcke durch das Klicken auf eine Zelle im KV Diagramm zusammengestellt werden
- Jeder farbliche Block erhält ein eigenes Eingabefeld, in dem dann der passende Boolesche Ausdruck zu diesem Block erstellt werden kann
- Es stehen zwei Felder für die Eingabe des KNF- und DNF-Ausdrucks zur Verfügung. Um diese zusammenzustellen, können die zuvor erwähnten farblichen Blöcke als Hilfsmittel verwendet werden
- Die gewählten farblichen Blöcke und die Eingabefelder der Booleschen Ausdrücke sollen auf ihre Richtigkeit geprüft werden können
- Bei Bedarf sollen Lösungen für die farblichen Blöcke, deren Boolesche Ausdrücke und sowohl für den KNF-Ausdruck als auch für den DNF-Ausdruck eingeblendet werden
- Bei Änderung des Booleschen Ausdrucks soll das KV Diagramm aktualisiert und alle dazugehörigen Eingabefelder zurückgesetzt werden
- Des Weiteren ist für ausländische Studenten ist die Möglichkeit eine Sprache auszuwählen sinnvoll

5.3 Nicht-funktionale Anforderungen

Die nicht-funktionalen Anforderungen ergänzen die funktionalen Anforderungen. Dort sind hauptsächlich Qualitäts- und Benutzbarkeitansprüche vorzufinden. Die einzelnen Ansprüche werden in Kategorien aufgeteilt. Zuerst werden jeweils die Kategorie erläutert und anschließend die Anforderungen aufgeführt. Dabei kann Bezug auf einige

Funktionen aus dem Praxisprojekt (s. Datenträger) genommen werden. Für die Kategorisierung werden folgende Qualitätseigenschaften der Norm DIN 66272 verwendet [Bächle (1996)].

5.3.1 Funktionalität

Die Funktionalität definiert die Fehlertoleranz der Applikation und das Vorhandensein von bestimmten Funktionen.
Funktionalität hat bei allen Funktionen eine hohe Wichtigkeit, welche mit der Berechnung des Ergebnisses zusammenhängen.

- Der Aufbau und das Auflösen des Binärbaums (s. Praxisprojekt) muss bei allen Booleschen Ausdrücken fehlerfrei erfolgen
- Aus einem Binärbaum soll ein dazu entsprechendes KV Diagramm erstellt werden. Die Zuordnung der Variablen erfolgt alphabetisch
- Während einer Validierung der Benutzereingaben bei den farblichen Blöcken, wird auf Übereinstimmung mit den markierten Zellen und nicht auf das Optimale geprüft
- Das optimale Ergebnis wird erst beim Minimierten Term (KNF und DNF) ersichtlich

Grundsätzlich sollten keine Bugs ode Abstürze vorhanden sein. Eine gewisse Fehlertoleranz ist lediglich bei optischen Genauigkeiten vorhanden, solange wichtige Funktionalitäten nicht beeinträchtigt werden. Optische Elemente können je nach Browser ein wenig von der Optik, Größe oder Position abweichen. Solche Fälle sollten jedoch so gut, wie möglich vorgebeugt werden.

5.3.2 Zuverlässigkeit

Zuverlässigkeit bedeutet die Aufrechterhaltung des Leistungsniveau der Software unter festgelegten Bedingungen und über einen festgelegten Zeitraum.
Höchste Priorität haben die Ergebnisse. Der Benutzer muss sich darauf verlassen können, dass Aufgaben auf dem Papier mit dieser Applikation validiert werden können. Es sollte immer Verlass darauf sein, dass das KV Diagramm zum Booleschen Ausdruck passt und dass durch diese Applikation der minimierte Term herausgefunden werden kann.

5.3.3 Benutzbarkeit

Die Benutzbarkeit ist der Aufwand, der zur Benutzung erforderlich ist und eine individuelle Beurteilung der Benutzung ermöglicht.
Die Bedienung des Tools sollte jedem intuitiv verständlich sein, der mit Boolescher Algebra vertraut ist. Die Eingabe von Sonderzeichen (Operatoren) erfolgt über Buttons. Das Kopieren, Ausschneiden und Einfügen von Ausdrücken sollte ebenfalls möglich sein (s. Praxisprojekt). Die gesamte Applikation sollte auf jedem Gerät bedient werden können.

5.3.4 Effizienz

Unter Effizienz wird das Verhältnis zwischen Leistungsniveau der Software und dem Umfang der eingesetzten Betriebsmittel unter festgelegten Bedingungen verstanden.
Der Benutzer sollte die Ergebnisse möglichst schnell erhalten. Die Ausführung aller Aufgaben sollte die im Browser festgelegte maximale Ausführungszeit nicht überschreiten (z.B. in Firefox sind es 10 Sekunden) [Daniel Kuhn (2013)].

5.3.5 Änderbarkeit

Die Änderbarkeit ist die Festlegung des Aufwands, der zur Durchführung vorgegebener Änderungen notwendig ist.
Die jetzige Funktionalität sollte durch spätere Erweiterungen nicht beeinträchtigt werden. Diese Applikation soll ohne großen Mehraufwand um zusätzliche Funktionen erweitert werden können.

5.3.6 Übertragbarkeit

Unter Übertragbarkeit wird die Eignung der Software vermerkt, von einer Umgebung in eine andere übertragen zu werden.
Eine Übertragung in ein anderes System ist nicht eingeplant. Die Applikation soll in einem modernen Browser laufen, der JavaScript unterstützt. Sollte eine Desktop oder Android Applikation gewünscht sein, so kann dies durch die Nutzung von Frameworks, wie *Electron*[1] oder *PhoneGap*[2] bewältigt werden, da diese ein HTML Frontend voraussetzen. Im Praxisprojekt (s. Datenträger, Kapitel 3.2.6) wurden bereits Browser-Nutzungsstatistiken aufgeführt und durch Summierung der Gesamtnutzungszahlen die Erkenntnis aufgestellt, dass es wichtig sei die Kompatibilität zu Google Chrome und Firefox zu behalten, um mindestens 80 Prozent des Browsermarktes abzudecken.

[1] Electron: `http://electron.atom.io/`
[2] PhoneGap: `http://phonegap.com/`

6 Technischer Einsatzkontext

In diesem Kapitel wird der Einsatz der einzelnen Systeme und Komponenten erläutert. Zuerst wird dazu ein Überblick gegeben und anschließend die verwendete Software beschrieben.

6.1 Überblick

Dieser Abschnitt zeigt in Abbildung 6.1 eine Übersicht zur technischen Nutzung und Zusammenhang aller notwendigen Systeme. Anschließend folgt eine Beschreibung zu dieser Abbildung.

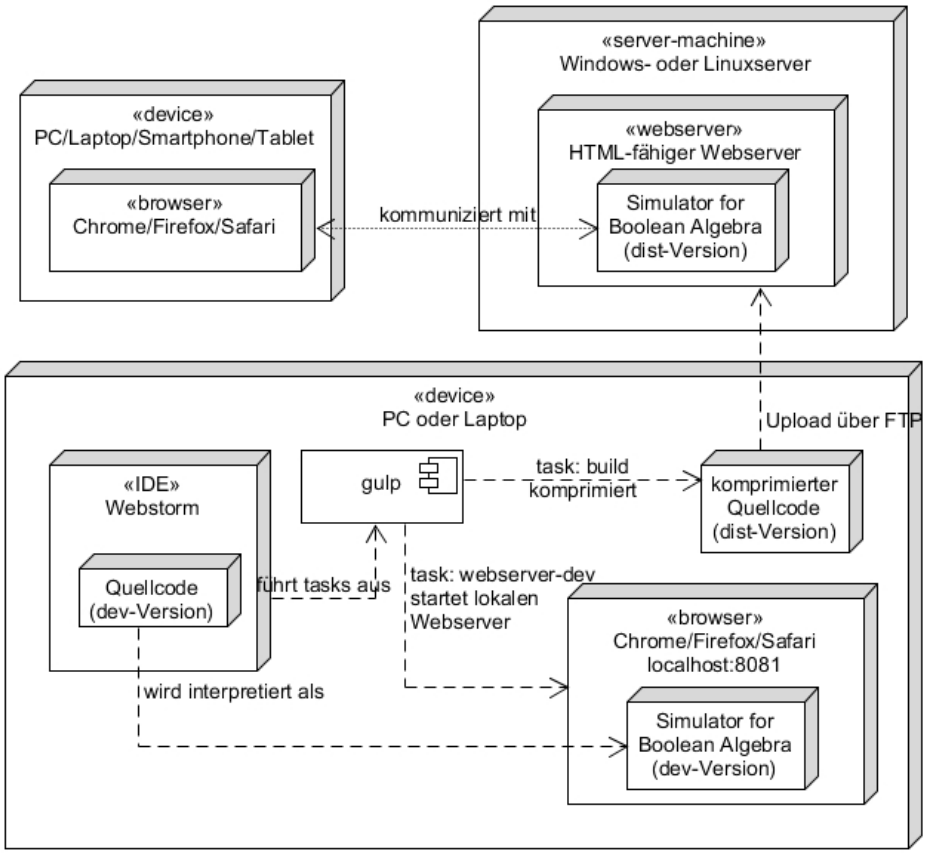


Abbildung 6.1: Überblick vom technischen Einsatz

Für den Überblick ist es wichtig den gesamten Einsatzkontext in der Abbildung 6.1 darzustellen. Dazu zählen dann auch die IDE und der Computer, auf dem die Applikation entwickelt wird.
Die Applikation wird nicht zu einer ausführbaren Datei kompiliert, sondern besteht aus HTML-, CSS- und JavaScript-Dateien, die vom Browser interpretiert werden. Es können sich also besonders bei Webapplikationen viele CSS- und JavaScript-Dateien ansammeln, die dann alle vom Browser abgerufen und ausgeführt werden sollen. Es ist wesentlich schneller eine einzige Datei abzurufen (besonders im Web), die zum Beispiel 1 Megabyte groß ist, als viele einzelne Dateien, die gemeinsam 1 Megabyte groß sind.
Deshalb wird im Buch *Performante Webanwendungen: Client- und serverseitige Techniken zur Performance-Optimierung* [von Daniel Kuhn (2013)] empfohlen, alle CSS- und JavaScript-Dateien zu komprimieren und dann in eine jeweils gemeinsame CSS- und JavaScript-Datei zusammenzuführen. Bei einer Komprimierung werden alle Kommentare, Leerzeichen und Zeilenumbrüche entfernt, sodass der Quellcode so kompakt, wie möglich in einer Zeile steht. Im Projekt, also auch in Abbildung 6.1, wird deshalb zwischen komprimierter Version (*dist* - Distribution) und nicht-komprimierter Version (*dev* - Development) unterschieden.
In der IDE wird nur an der *dev*-Version gearbeitet. Diese wird auf einem lokalen Webserver betrieben und ist über den Browser auf der URL *http://localhost:8081/* ausführbar. Da der Quellcode in seiner ursprünglichen Form, also auch in einzelne Dateien aufgeteilt ist, können hier besser Fehler entdeckt werden. Um dies zu ermöglichen, wurde mit Gulp (Beschreibung vorzufinden in Tabelle 6.2) der Task *webserver-dev* entwickelt. Dieser startet einen lokalen Webserver und bindet eine *Livereload* (Beschreibung vorzufinden in Tabelle 6.2) von Gulp ein. Zur Komprimierung und zum Konkatenieren der CSS- und JavaScript-Dateien wurde der sog. *build*-Task entwickelt. Dieser führt das Minimieren der Dateien durch und erstellt in einem neuen Ordner die gesamte komprimierte Version der Applikation, die dann ohne Weiteres auf einem Webserver betrieben werden kann. Um die komprimierte Version lokal zu testen, wurde der Task *webserver-dist* erstellt. Dieser funktioniert wie *webserver-dev*. Der einzige Unterschied der Tasks ist, dass *webserver-dist* die komprimierte Applikation verwendet und auf der URL *http://localhost:8080/* lauscht.

6.2 Entwicklungsumgebung und verwendete Software

Alle verwendeten Programme und Frameworks werden in Tabellen alphabetisch aufgelistet und in deren Nutzungszweck beschrieben. Tabelle 6.1 enthält Software, die als eigene Applikation gilt und installiert werden sollte. Des Weiteren sind einige Frameworks notwendig. Hierbei wird von Backend und Frontend Frameworks unterschieden. Backend Frameworks (siehe Tabelle 6.2) unterstützen aktiv die Entwicklung und sind lediglich für den Entwickler verfügbar. Wohingegen Frontend Frameworks (in Tabelle 6.3) in der Applikation eingesetzt wurden und mit der Applikation ausgeliefert werden.

Name	**Beschreibung**
Apple Safari[1]	Webkit Browser von Apple. Wurde dazu verwendet, um Abweichungen seitens Apple Browser zu erkennen. (Version 5.1.7 Windows und Version 8.0 Mac OS X)
Google Chrome[2]	Webkit Browser von Google. Wurde für die Entwicklung eingesetzt. Die Entwicklertools halfen beim Debugging. (Version 55.0.2883.87 m Windows und Version 55.0.2883.91 Android)
JetBrains WebStorm 2016[3]	Entwicklungsumgebung für HTML5, CSS3 und JavaScript (Version 2016.2.4, student license)
Mozilla Firefox[4]	Moz-Browser von Mozilla. Wurde zum Testen auf Kompatibilität von Moz-Engine getestet. (Version 51.0.1)
NodeJS[5]	Runtime Umgebung basierend auf Chrome's V8 JavaScript Engine, die dazu verwendet wird, um Gulp und Bower zu verwenden (Version 6.9.1)

Tabelle 6.1: Verwendete Programme

[1] Apple Safari: `http://www.apple.com/de/safari/`
[2] Google Chrome: `https://www.google.de/chrome/browser/desktop/`
[3] JetBrains WebStorm: `https://www.jetbrains.com/webstorm/`
[4] Mozilla Firefox: `https://www.mozilla.org/de/firefox/new/`
[5] NodeJS: `https://nodejs.org/en/`

Name	Beschreibung
Bower[6]	Ein Package-Manager, der das einfache Installieren und Aktualisieren von Programmbibliotheken und Frameworks ermöglicht. Dieser wird für die Frontend Frameworks verwendet. (Version 1.8.0)
Gulp[7]	Ein Task-Runner, um Workflow-Tasks im Webentwicklungsprozess zu automatisieren. Dieser wird verwendet, um einen lokalen Webserver mit Livereload-Funktion zu starten und um den Quellcode in eine kompakte Version zu komprimieren. (Version 3.9.1)
(Gulp)-Cssmin[8]	Ein Submodul von Gulp, dass für die Komprimierung von CSS-Dateien zuständig ist. (Version 0.1.7)
(Gulp)-Livereload[9]	Ein Submodul von Gulp, dass auf Änderung im Quellcode achtet und diese umgehend (live) in der Applikation einbringt. (Version 3.8.1)
(Gulp)-Uglify[10]	Ein Submodul von Gulp, dass für die Komprimierung von JavaScript-Dateien zuständig ist. (Version 1.5.4)
(Gulp)-Useref[11]	Ein Submodul von Gulp, dass für die Konkatenation von Dateien dient. (Version 3.1.2)

Tabelle 6.2: Backend Frameworks

[6] Bower: `https://bower.io/`
[7] Gulp: `http://gulpjs.com/`
[8] Cssmin: `https://www.npmjs.com/package/gulp-cssmin`
[9] Livereload: `https://www.npmjs.com/package/gulp-livereload`
[10] Uglify: `https://www.npmjs.com/package/gulp-uglify`
[11] Useref: `https://www.npmjs.com/package/gulp-useref`

Name	Beschreibung
AngularJS[12]	Clientseitiges Framework für die Erstellung von Single-page-Webanwendungen. Wird dafür verwendet, um die Modularität der Applikation einzuhalten. (Version 1.4.14)
AngularJS-Route[13]	Eine AngularJS Komponente für das Routing von URLs. Wird dafür verwendet, um ein eigenes View für diese Lernapplikation anzubieten, damit diese leichter in eine andere AngularJS Applikation implementiert werden kann. (Version 1.4.14)
AngularJS-Translate[14]	Sprachen können in JSON-Dateien ausgelagert und im HTML-View verwendet werden. (Version 2.11.1)
Bootstrap[15]	Frontend Framework für die Entwicklung von Responsive Weblösungen. Es wurde dafür verwendet, um das HTML Layout so zu gestalten, sodass es auf jedem Gerät gut sichtbar dargestellt wird. (Version 3.3.7)
EaselJS[16]	Ein HTML5 Canvas Framework für die einfache Erstellung von grafischen Elementen mit der Möglichkeit auf Interaktion mit Eingabegeräten und Steuerung durch Ereignisse. (Version 0.8.2)
ExtendJS[17]	Vereinfachung von Vererbung bei JavaScript. (Version 0.2.2)
Flag-Icon-Css[18]	Sammlung von Flaggen-Icons, benutzbar durch CSS-Klassen. (Version 2.7.0)
Font-Awesome[19]	Sammlung von Icons für das User Interface, benutzbar durch CSS-Klassen. (Version 4.7.0)
jQuery[20]	Stellt vereinfachte Funktionen für die DOM[21] Navigation zur Verfügung. (Version 3.1.1)

[12] AngularJS: https://angularjs.org/
[13] AngularJS-Route: https://docs.angularjs.org/api/ngRoute
[14] AngularJS-Translate: https://github.com/angular-translate/angular-translate
[15] Bootstrap: http://getbootstrap.com/
[16] EaselJS: http://www.createjs.com/easeljs
[17] ExtendJS: http://extendjs.org/
[18] Flag-Icon-Css: http://flag-icon-css.lip.is/
[19] Font-Awesome: http://fontawesome.io/
[20] jQuery: https://jquery.com/
[21] DOM: https://de.wikipedia.org/wiki/Document_Object_Model

JSON-Formatter[22]	Eine AngularJS Direktive für das Auslesen von JSON-Dateien. Wird von AngularJS-Translate für das Auslesen der Sprachdaten verwendet. (Version 0.5.0)
Modernizr[23]	Ermöglicht die Erkennung von HTML5- und CSS3-Features in verschiedenen Browsern. Sollte eine Funktion bei einem Browser nicht akzeptiert werden, so wird diese durch Workarounds nachgebildet. (Version 2.8.3)

Tabelle 6.3: Frontend Frameworks

6.3 Komprimierung mit Hilfe von Gulp

Das Submodul *(Gulp)-Useref* verpackt mehrere Dateien in eine. Um zu entscheiden, welche Dateien konkateniert werden sollen, ist die Definition des Dateityps und gewünschten Dateinamen notwendig. Dies erfolgt über einen HTML-Kommentar (Bspw. *<–build:js js/app.min.js–>*). Darunter folgen die eingebundenen Dateien. Ein weiterer HTML-Kommentar (*<– endbuild –>*) beendet dann diese Aufzählung. Im Task *build* wird die Useref-Funktion verwendet. Diese sucht sich den einleitenden Kommentar raus und fasst alle darauf folgenden Dateien zusammen. Sollte die Useref-Funktion nicht verwendet werden (zum Beispiel in Task *webserver-dev*), dann werden die Dateien nicht zusammengefasst und der Kommentar lediglich als HTML-Kommentar betrachtet. Ein Beispiel ist in Abbildung 6.2 zu sehen.

[22] JSON-Formatter: `https://github.com/mohsen1/json-formatter`
[23] Modernizr: `https://modernizr.com/docs`

```
<!-- build:js js/app.min.js -->
<script src="js/constants.js"></script>
<script src="js/utils.js"></script>
<script src="js/ext.js"></script>

<script src="js/ColorGenerator.js"></script>

<script src="js/BoolAlgebra/BADomain.js"></script>
<script src="js/BoolAlgebra/BAGroup.js"></script>
<script src="js/BoolAlgebra/BANode.js"></script>
<script src="js/BoolAlgebra/BATable.js"></script>
<script src="js/BoolAlgebra/BAExpression.js"></script>

<script src="js/Canvas/CanvasInterface.js"></script>
<script src="js/Canvas/EaselInterface.js"></script>

<script src="js/KV/ColorPathMap.js"></script>
<script src="js/KV/ColorPathLayer.js"></script>

<script src="js/KV/KVCell.js"></script>
<script src="js/KV/KVRow.js"></script>
<script src="js/KV/KVBlock.js"></script>
<script src="js/KV/KVDiagram.js"></script>
<script src="js/KV/BAKV.js"></script>
<script src="js/KV/KVExprCompare.js"></script>

<script src="js/KV/KVReflectingBlock.js"></script>
<script src="js/KV/KVReflectingSearch.js"></script>

<script src="js/app.js"></script>
<!-- endbuild -->
```

Abbildung 6.2: Beispiel für Useref

6.4 Gulp-Tasks innerhalb der WebStorm IDE

Die in Abschnitt 6.1 erwähnten Gulp-Tasks können über die Eingabeaufforderung ausgeführt werden, sofern NodeJS installiert ist. Der *build*-Task kann durch Eingabe von *gulp build* in die Eingabeaufforderung ausgeführt in werden. Dabei ist jedoch wichtig, dass der aktuelle Pfad in der Eingabeaufforderung das Root-Verzeichnis der Projektmappe ist (siehe Abbildung 6.3). Bei WebStorm besteht die Möglichkeit den Aufwand mit der Eingabeaufforderung zu vermeiden. Im oberen rechten Bereich auf der Benutzeroberfläche von WebStorm gibt es einen grünen Play-Button, auf den Funktionen gebunden werden können. Dort wird dann Gulp als Service ausgewählt und lediglich der gewünschte Task eingetragen und benannt. In Abbildung 6.4 wird der Task *webserver-*

dev auf den Namen *Run Dev* gebunden. Dieser erscheint dann neben dem Play-Button (siehe Abbildung 6.5) und kann durch die Betätigung dieses Buttons ausgeführt werden.

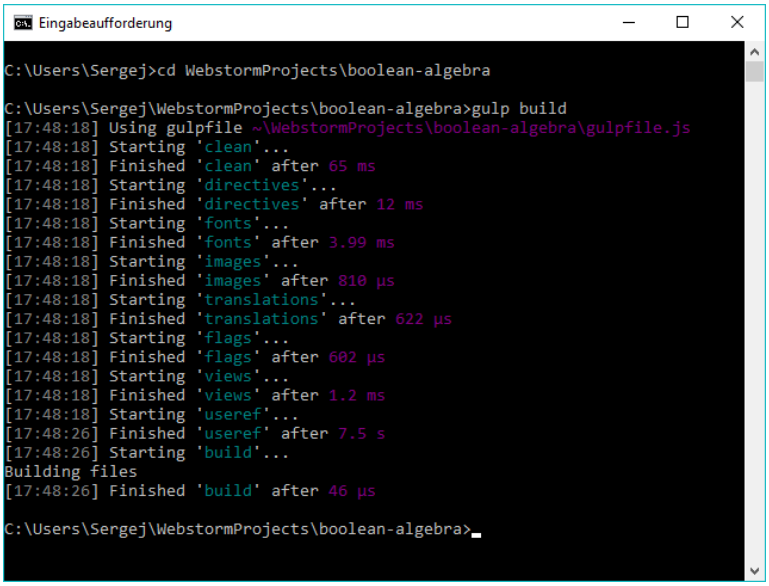


Abbildung 6.3: Eingabeaufforderung: gulp build

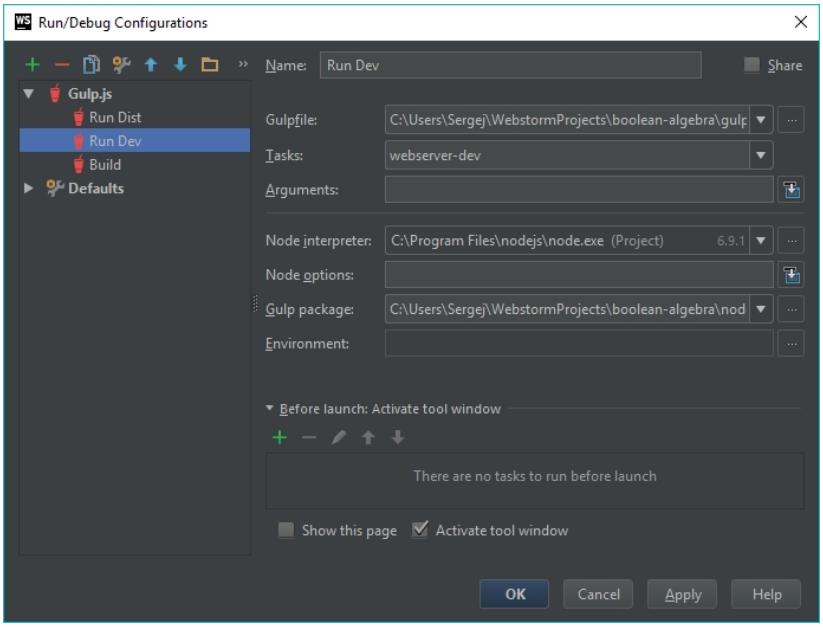


Abbildung 6.4: WebStorm Play-Button Konfiguration

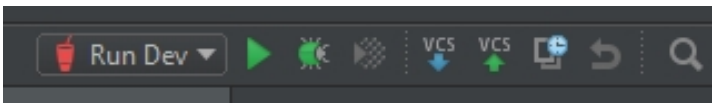


Abbildung 6.5: WebStorm Play-Button - Run Dev

6.5 Entscheidung für das EaselJS Framework

Für die Darstellung des KV Diagramms wurde ein HTML5 Canvas[24] Framework gesucht, welches folgende Kriterien erfüllt:

(A) Möglichkeit Shapes zu erstellen

(B) Keine große Einarbeitung notwendig

(C) Eigene Elemente aus Shapes möglichst unkompliziert zusammenstellen und durch Ereignisse steuern

(D) Zeichnen von Linien und abgerundeten Rechtecken sollte möglich sein.

(E) Möglichst wenig Overhead (Funktionen die nie zum Einsatz kommen)

Es wurden einige HTML5 Canvas Frameworks ausprobiert und mit den Kriterien abgewägt. **Legende:** J = Ja, N = Nein, T = Teilweise

Framework	**A**	**B**	**C**	**D**	**E**
EaselJS[25]	J	J	J	J	J
Fabric.js[26]	J	J	N	J	N
Paper.js[27]	Ja	N	T	J	N
pixijs.com[28]	J	N	J	J	N

Tabelle 6.4: Vergleich der HTML5 Canvas Frameworks

Das einzige Framework, das alle Kriterien erfüllt hat ist EaselJS. Wegen seiner Einfachheit kann schnell das gewünschte Ergebnis erzielt werden. Die Ereignisse für Shapes sind nach den Standardereignissen von JavaScript[29] benannt. Die Shapes lassen sich einfach mit *Container*[30] zusammensetzen auf den dann Ereignisse gebunden werden können.

[24] HTML5 Canvas: `http://www.w3schools.com/html/html5_canvas.asp`
[25] EaselJS: `http://www.createjs.com/easeljs`
[26] Fabric.js: `http://fabricjs.com/`
[27] Paper.js: `http://paperjs.org/tutorials/getting-started/`
[28] pixijs.com: `http://www.pixijs.com/`
[29] JavaScript Events: `http://www.w3schools.com/js/js_events.asp`
[30] EaselJS Container: `http://www.createjs.com/docs/easeljs/classes/Container.html`

7 Implementierung

Dieses Kapitel befasst sich mit der Implementierung der Applikation.
In Kapitel 7.1 wird die Systemarchitektur mit UML-Diagrammen vorgestellt. Zuerst werden in den Unterkapiteln 7.1.1 und 7.1.2 der Systementwurf in Form von Komponentendiagrammen dargestellt. Danach folgt in Unterkapitel 7.1.3 ein Klassendiagramm mit einer tabellarischen Beschreibung zu den Klassen, gefolgt von Aktivitätsdiagrammen (s. Kapitel 7.1.4) für die Benutzerinteraktion mit dem KV Diagramm und Pseudocode (s. Kapitel 7.2) für den Such-Algorithmus und Blockfärbung. Zum Abschluss wird in Kapitel 7.3 die Umsetzung der Ergebnisprüfung erläutert.

7.1 Systemarchitektur

Durch UML-Diagramme wird eine programmiersprachenunabhängige Sichtweise auf die Umsetzung der Applikation ermöglicht. Die Diagramme werden, um den Rahmen nicht zu sprengen, auf das Wesentliche reduziert.
Zur Modellierung der UML Diagramme wurde das freie Tool UMLet[1] verwendet.

7.1.1 Gesamtübersicht als Komponentendiagramm

Im Praxisprojekt (s. Datenträger, Kapitel 5.1.1) wurde die Modularität durch AngularJS in Bezug auf die Lernplattform beschrieben. Dort wurde diese durch ein Komponentendiagramm verdeutlicht. Dieses wird, um es in den Vordergrund zu rufen, in Abbildung 7.1 angezeigt. Jede beliebige AngularJS oder Webseite kann die erstellten Lernmodule ohne großen Aufwand implementieren.

[1] UMLet: `http://www.umlet.com/`

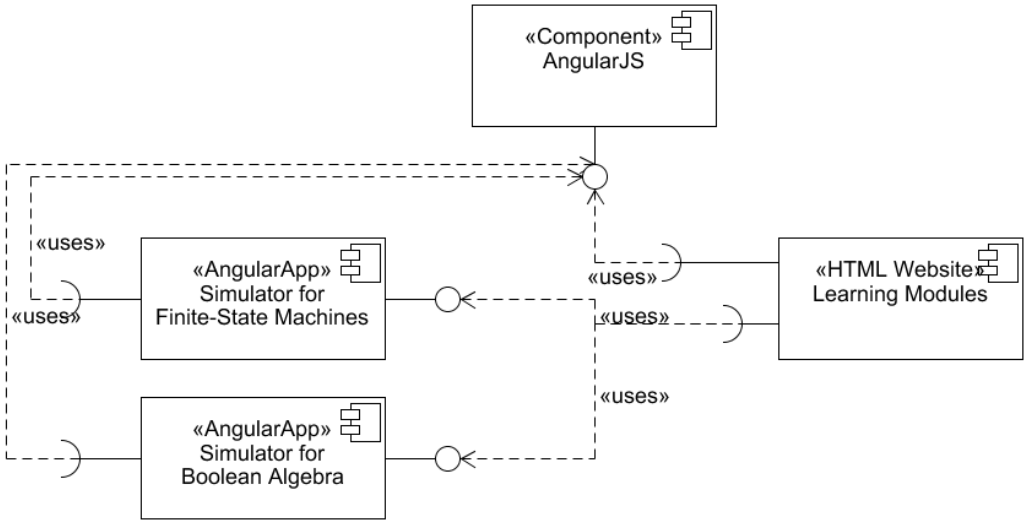


Abbildung 7.1: Komponentendiagramm für die Übersicht

7.1.2 Komponentediagramm der Applikation

Die in Kapitel 7.1.1 erwähnte Unabhängigkeit der Teilmodule kommt in diesem Projekt zur Geltung. Das im Praxisprojekt Kapitel 5.1.2 eingesetzte Komponentendiagramm der AngularJS Module wurde problemlos um drei weitere Direktiven erweitert. In diesem Kapitel wird nur auf diese drei Direktiven eingegangen. Die Beschreibung der weiteren Direktiven ist der Abbildung im Praxisprojekt zu entnehmen. Das angepasste Komponentendiagramm ist in Abbildung 7.1 zu sehen.

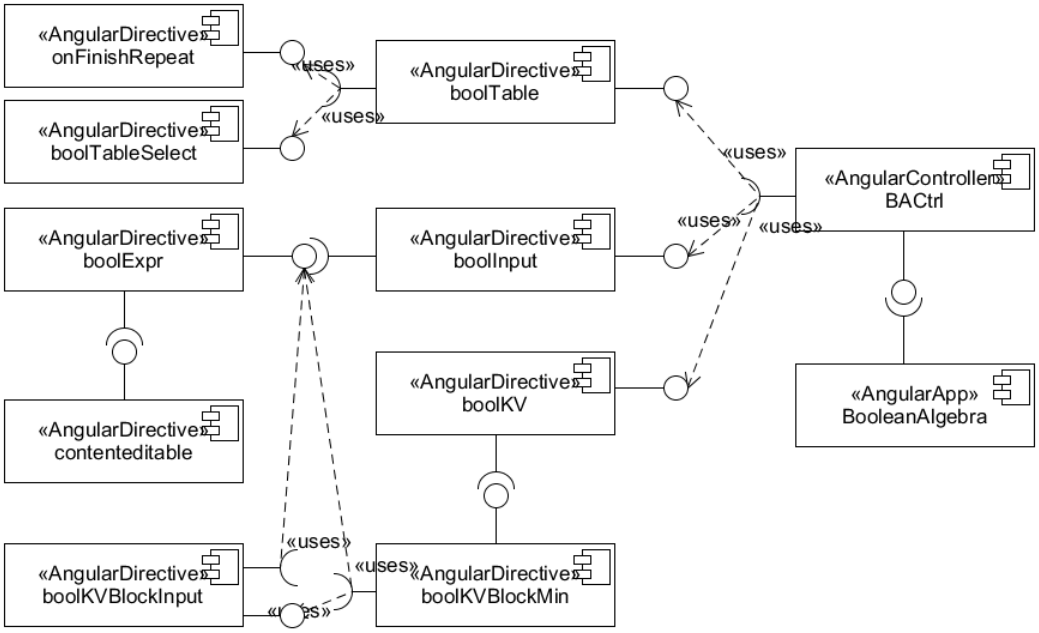


Abbildung 7.2: Komponentendiagramm - AngularJS Module

Die wichtigste neue Direktive ist *boolKV*. Das ist die Hauptdirektive der KV Diagramm Funktionalitäten, die im Grunde den Rahmen für das KV Diagramm legt. Sie implementiert *boolKVBlockMin*. Die Direktive *boolKVBlockMin* enthält die Struktur und Funktionalität der Eingabefelder für farbliche Blöcke, der KNF und der DNF. Da ein Eingabefeld für einen farblichen Block sich auf ausgewählte Zellen bezieht und eine Farbe anstelle einer Bezeichnung enthält, wurde hierfür eine weitere Direktive entwickelt, die *boolExpr* erweitert.

7.1.3 Klassendiagramm

In Abbildung 7.3 ist das Klassendiagramm für die Übersicht nur auf die Klassennamen reduziert. Das vollständige Klassendiagramm kann im Anhang betrachtet werden. Dieses Klassendiagramm knüpft an das im Praxisprojekt (s. Datenträger, Kapitel 5.1.3) an. Es wurden hier die Klassen ausgelassen, die für die zu implementierenden Erweiterungen irrelevant sind. Die einzige aus dem Klassendiagramm des Praxisprojekts wiederverwendete Klasse ist *BAExpression*. Neben eigenen Klassen wurden hier auch ein paar aus der *EaselJS*-Bibliothek verwendet. Eine weitere Ausnahme ist die Entität *HTML5 Canvas*. Diese repräsentiert die in HTML5 zur Verfügung gestellte Leinwand für grafische Objekte. Nach der Abbildung 7.3 folgen tabellarisch die einzelnen Erläuterungen zu den Klassen.

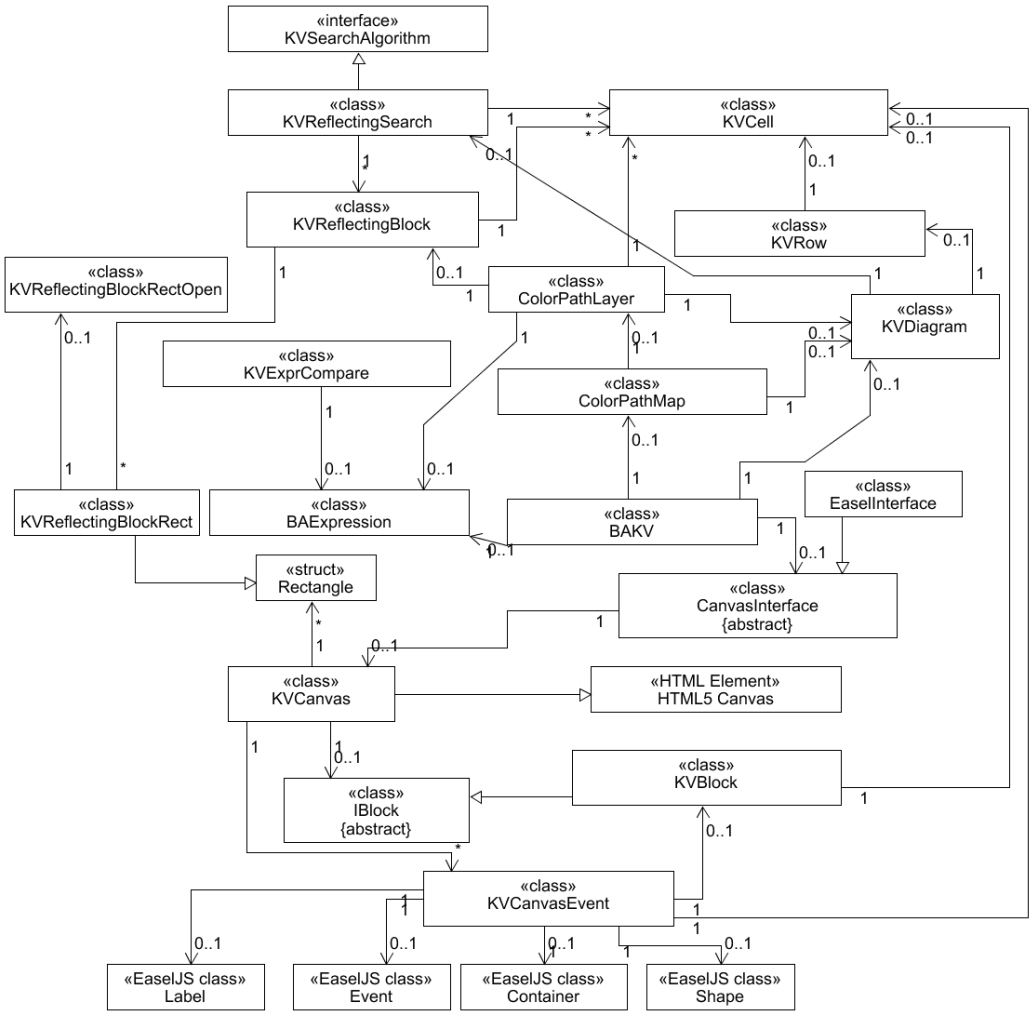


Abbildung 7.3: Klassendiagramm der Applikation (gekürzt)

Um Einheit zu schaffen werden die Klassen bei der Beschreibung in Verantwortungsgruppen kategorisiert. Hierbei wurden nur die Klassen beschrieben, die in diesem Projekt selber entwickelt worden sind. Das bedeutet, dass sowohl *HTML5 Canvas*, *Label*, *Event*, *Container*, *Shape*, *Rectangle* als auch die bereits im Praxisprojekt beschriebene Klasse *BAExpression* wegfallen.

Suchalgorithmus

In diesem Abschnitt werden Klassen in der Tabelle 7.1 aufgeführt, die zum Such-Algorithmus gehören, der in Kapitel 4.4.3 konzipiert wurde.

Name	**Beschreibung**
KVSearchAlgorithm	Eine Schnittstelle für Suchalgorithmen im KV Diagramm. Neue Suchalgorithmen können den aktuellen problemlos ersetzen, wenn sie die Form der Schnittstelle einhalten.
KVReflectingSearch	Eine Klasse für die Implementierung des in Abschnitt 4.4.3 geplanten Suchalgorithmus. Sie erbt die Klasse *KVSearchAlgorithm*.
KVReflectingBlock	Datenstruktur für die Handhabung eines farblichen Blocks im Suchalgorithmus.

Tabelle 7.1: Klassen für den Suchalgorithmus

Datenstruktur und Funktion

Für den Aufbau des KV Diagramms und dessen Interaktionsfunktionen wurden folgende Klassen in der Tabelle 7.2 aufgezählt und beschrieben:

Name	**Beschreibung**
KVCell	Repräsentiert eine Zelle im KV Diagramm.
KVRow	Enthält Methoden für eine bequemere Verwaltung von Zellen in einer Zeile.
KVDiagram	Repräsentiert das KV Diagramm an sich und enthält somit Funktionen zum Aufbau und Handhabung eines KV Diagramms.
ColorPathLayer	Stellt eine Ebene für eine Farbe dar, in der sich vom Benutzer erstellte farbliche Blöcke dieser Farbe befinden.
ColorPathMap	Zuständig für die Verwaltung von *ColorPathLayer* Instanzen.
KVExprCompare	Ist für das Vergleichen von Booleschen Ausdrücken zuständig, wie zum Beispiel beim KNF und DNF.
BAKV	Dient als Schnittstelle zwischen der *boolKV* Direktive, der KV Diagramm Datenstruktur und des *KVCanvas*. Es stellt somit Funktionen zur Verfügung, die von der Benutzeroberflache über die Direktive abgerufen werden.

Tabelle 7.2: Klassen für die Datenstruktur

Grafische Darstellung

Die Klassen in Tabelle 7.3 sind für das Darstellen und Interagieren mit den grafischen Elementen des KV Diagramms zuständig.

Name	**Beschreibung**
CanvasInterface	Bietet eine Grundstruktur für die Verwaltung eines HTML5 Canvas Frameworks. So kann das aktuelle Framework bei Bedarf ohne großen Mehraufwand durch ein anderes ersetzt werden.
EaselInterface	Eine Klasse, die intern ein HTML5 Canvas mit grafischen Objekten, wie Labels und Buttons über das EaselJS Framework (s. Tabelle 6.3) befüllt.
KVCanvas	Ein HTML5 Canvas, das durch *CanvasInterface* um Funktionen und Ereignisse erweitert wurde.
IBlock	Repräsentiert ein grafisches Rechteck im *KV Canvas*. Damit konnten die Variablen Labels korrekt konstruiert und positioniert werden.
KVBlock	Basiert auf *IBlock* und repräsentiert das grafische Objekt einer Zelle im KV Diagramm.
KVCanvasEvent	Ist ein Ereignisträger von Interaktionsereignissen auf dem KV Diagramm.
KVReflectingBlockRect	Informationsträger für Position und offenen Seiten (*KVReflectingBlockRectOpen*) eines farblichen Blocks.
KVReflectingBlockRectOpen	Eine Struktur für die Information, wo ein farblicher Block offen ist.

Tabelle 7.3: Klassen für die grafische Darstellung

7.1.4 Aktivitätsdiagramme

Die Aktivitätsdiagramme, die in nachfolgenden Abschnitten vorgestellt werden, dienen zur Verdeutlichung wichtiger Abläufe. Einer dieser Abläufe stellt die Interaktion mit dem KV Diagramm und die darauffolgende Generierung der farblichen Rechtecke eines Blocks dar. Für zwei der Teilabläufe wurden in Kapitel 7.2 Pseudocodes erstellt.

Interaktion zwischen Benutzer und KV Diagramm

Der Ablauf beginnt beim Aktivitätsdiagramm in Abbildung 7.4.
Der Benutzer wählt eine Farbe und danach eine Zelle aus, die dann an *Analysiere Farbebene* (s. Abbildung 7.5) übergeben werden. Die Blöcke, die zurückgeliefert werden, werden an den Algorithmus 3 *ColorRects-Algorithmus* übergeben, dessen Ergebnis dann im Canvas gerendert wird.

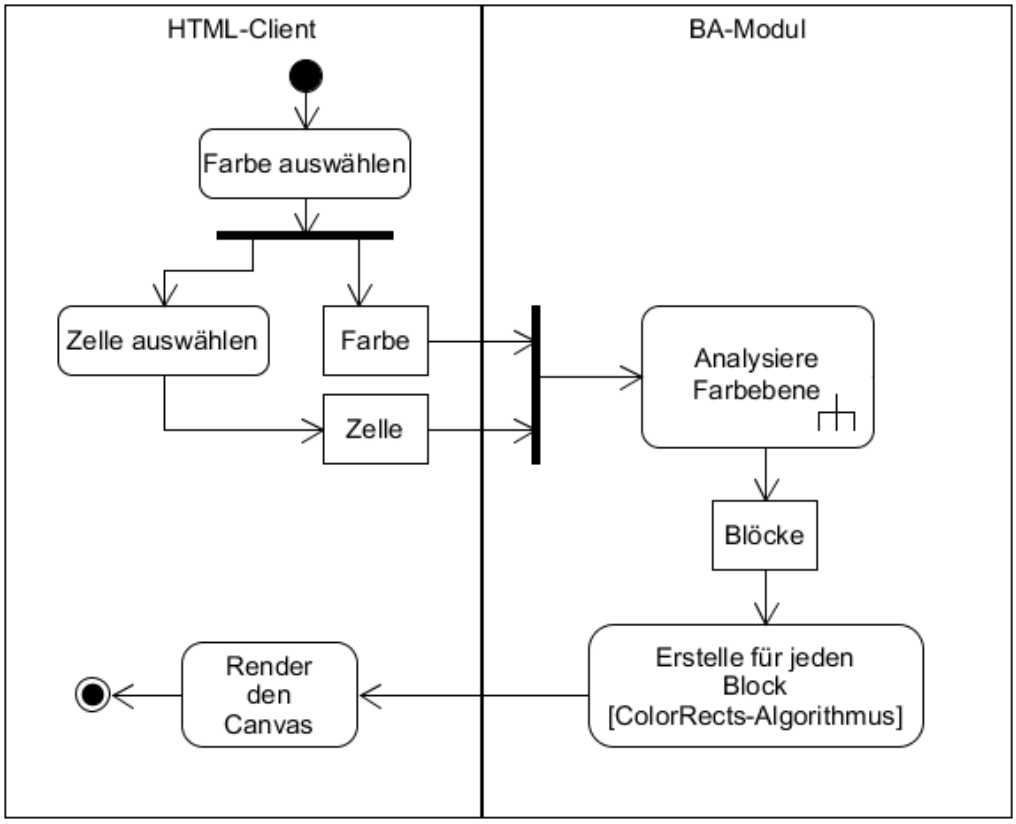


Abbildung 7.4: Aktivitätsdiagramm der Interaktion

Analyse der Farbebenen

In Abbildung 7.5 beginnt die eine Analyse für die ausgewählte Farbe.
Es wird eine Liste (*ColorPathMap*) an Ebenen (*ColorPathLayer*) für jede Farbe gehalten. Wenn eine Ebene noch nicht existiert wird diese erstellt. Die ausgewählte Zelle wird dieser Ebene hinzugefügt, falls sie dort noch nicht vorhanden ist. Ist sie vorhanden, dann wird sie wieder entfernt. Dies ähnelt der Funktionsweise eines Schalters. Anschließend wird der *KV-Reflecting-Search* Algorithmus (s. Algorithmus 1) ausgeführt, der dann Blöcke zurückliefert.

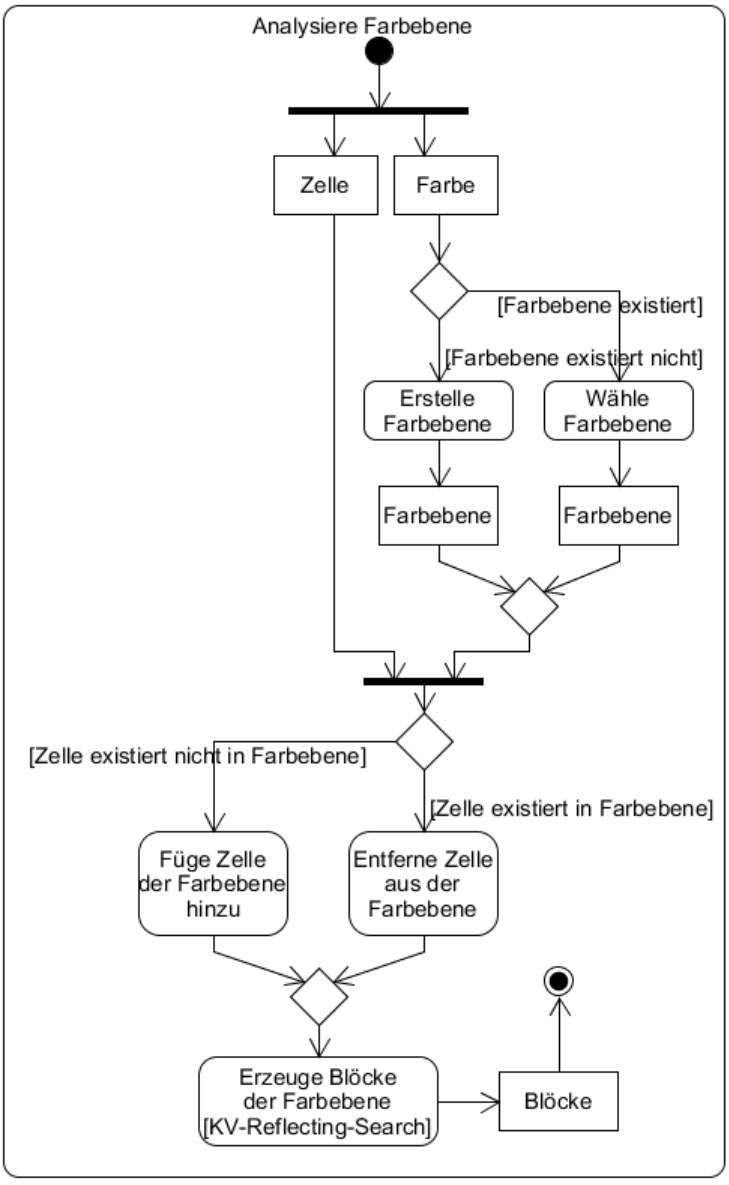


Abbildung 7.5: Aktivitätsdiagramm für Farbebenenanalyse

7.2 Pseudocodes

In Abschnitt 7.2.1 wird der Such-Algorithmus, der in Kapitel 4.4.3 entworfen und in Abbildung 7.5) verwendet wurde in Form eines Pseudocodes dargestellt. Zusätzlich folgt in Abschnitt 7.2.2 noch ein Pseudocode vom Algorithmus, der in Abbildung 7.4 eingesetzt wurde. Zu jedem Algorithmus steht auch eine Beschreibung des Vorgangs.

7.2.1 KV-Reflecting-Search-Algorithmus

Der in Kapitel 4.4.3 geplante Suchalgorithmus für größtmögliche Blöcke wurde erfolgreich umgesetzt. In diesem Abschnitt wird der dazu erstellte Pseudocode vorgestellt. Zuerst erfolgt jedoch eine Erläuterung zum Vorgang. Da JavaScript teilweise spezielle Herangehensweisen hat, könnte der Pseudocode von der Syntax her teilweise vom echten Code abweichen. Der Vorgang bleibt jedoch identisch. Im eigentlichen Code ist der Algorithmus in einer Klasse ausgelagert. Der Algorithmus erweitert ein Feld (Array) von Zellen und liefert dann ein Feld an gefundenen Blöcken zurück.
Die Funktion *Search* in Algorithmus 1 dient als Einstiegspunkt im Algorithmus. Der bevorzugte Wert und das Zellenfeld, mit welchem gearbeitet wird, werden für den Start übergeben. Als ersten Schritt wird eine leere *KVReflectingBlock* Liste (*blocks*) für später erstellt. In *Init* werden alle Zellen auf *unbesucht* gesetzt. Danach wird durch jede Zelle iteriert. Die aktuelle Zelle dieser Iteration wird dann als Startpunkt bei einer neuen *KVReflectingBlock*-Instanz verwendet. Die Blockinstanz repräsentiert einen farblichen Block. Dieser Block wird durch *Expand* auf seine maximale Größe gebracht. Anschließend wird der Block der am Anfang erstellten Blockliste (*blocks*) hinzugefügt, die dann am Ende als Ergebnis der Suche fungiert. Sofern eine Zelle bereits besucht wurde oder einen unerwünschten Wert enthält, wird diese bei der Iteration übersprungen. Die Prozedur *Expand* führt *Reflect*, solange *true* zurückgeliefert wird, aus. Ist die Schleife mit *Reflect* beendet, werden alle Zellen, die sich im aktuellen Block befinden als *besucht* markiert. Dies sorgt vor allem dafür, dass bei der Iteration in *Search* nicht alle Zellen durchlaufen werden. Bei der Prozedur *Reflect* werden jeweils die Methoden *ReflectRight*, *ReflectDown*, *ReflectLeft* und *ReflectUp* geprüft. Zuerst werden die inneren Zellen (Übergabewert *false*) und danach die äußeren Zellen (Übergabewert *true*) in jede Richtung betrachtet. Dies geschieht solange eine dieser Methoden ein positives Ergebnis zurückliefert.
Die Methoden *ReflectRight* und *ReflectDown* für den *KVReflectingBlock* wurden in Algorithmus 2 ausgelagert. *ReflectLeft* und *ReflectUp* hingegen wurden ausgelassen, weil sie jeweils den anderen beiden ähneln. Der einzige Unterschied ist die Richtung, die dabei durchlaufen wird. In *ReflectRight* wird für jede Zeile aus dem Zellenfeld die erste und letzte Zelle zwischengespeichert. Beginnend bei der letzten Zelle (*last*) wird so oft

der rechte Nachbar *next.right* als die nächste zu betrachtende Zelle *next* ausgewählt, wie der expandierende Block zu dem Zeitpunkt breit ist (*this.getWidth()*). Der Gesamtvorgang dieser Methode wird unterbrochen und als ungültig erklärt, wenn *next* die erste Zelle (*first*) erreicht, über den Rand hinausläuft (obwohl *throughWall* negativ ist) oder dessen Wert ungültig ist. Sollte keine dieser Abbruchkriterien zutreffen, so wird *next* in einer temporären Zeile (*collectionRow*) gesammelt. Sobald jede nach rechts erwartete Zelle durchlaufen wurde, wird die temporäre Zeile in der Liste *collection* gespeichert. Sollte der soeben beschriebene Durchlauf für jede Zeile im Block erfolgreich sein, so wird das Zellenfeld des aktuell expandierenden Blocks um *collection* erweitert. Dies hat einen performanten Gesamtdurchlauf zur Folge, da nur das durchlaufen wird, was möglich ist und Zellen vorwiegend durch Referenzen besucht werden.

Algoritmus 1 KV-Reflecting-Search-Algorithmus

```
1: function SEARCH(cellField, value)
2:     blocks = new List<KVReflectingBlock>()
3:     Init(cellField)
4:     for each KVCell cell in cellField do
5:         if cell.visited == true or cell.value != value then
6:             continue
7:         end if
8:         block = new KVReflectingBlock(cell, cellField)
9:         Expand(block)
10:        blocks.push(block)
11:    end for
12:    return blocks
13: end function
```

```
1: procedure INIT(cellField)
2:     for each KVCell cell in cellField do
3:         cell.visited = false
4:     end for
5: end procedure
```

```
1: procedure EXPAND(block)
2:     while Reflect(block) do
3:     end while
4:     for each KVCell cell in block.cells do
5:         cell.visited = true
6:     end for
7: end procedure
```

```
1: procedure REFLECT(block)
2:     if block.ReflectRight(false) then
3:         return true
4:     else if block.ReflectDown(false) then
5:         return true
6:     else if block.ReflectLeft(false) then
7:         return true
8:     else if block.ReflectUp(false) then
9:         return true
10:    else if block.ReflectRight(true) then
11:        return true
12:    else if block.ReflectDown(true) then
13:        return true
14:    else if block.ReflectLeft(true) then
15:        return true
16:    else if block.ReflectUp(true) then
17:        return true
18:    end if
19:    return false
20: end procedure
```

Algoritmus 2 KV-Reflecting-Search-Algorithmus: KVReflectingBlock

```
1: function ReflectRight(througWall)
2:     collection = new List<KVRow>()
3:     for r = 0; r < this.getHeight(); r++ do
4:         row = this.cells[r]
5:         first = row[0]
6:         last = row[row.length - 1]
7:         next = last
8:         collectionRow = new List<KVCell>()
9:         for c = 0; c < this.getWidth(); c++ do
10:            next = next.right
11:            if next.n < last.n and !throughWall) then
12:                return false
13:            end if
14:            if next.equals(first) or next.value != last.value then
15:                return false
16:            end if
17:            collectionRow.push(next)
18:        end for
19:        collection.push(collectionRow)
20:    end for
21:    this.Merge(this.cells, collection)
22: end function
```

```
1: function ReflectDown(througWall)
2:     lastRow = this.cells[this.cells.length - 1]
3:     firstRow = this.cells[0]
4:     collection = new List<KVRow>()
5:     for c = 0; c < this.getWidth(); c++ do
6:         first = firstRow[c]
7:         last = lastRow[c]
8:         next = last
9:         for r = 0; r < this.getHeight(); r++ do
10:            next = next.bottom
11:            if next.n < last.n and !throughWall then
12:                return false
13:            end if
14:            if next.equals(first) or next.value != last.value then
15:                return false
16:            end if
17:            if collection.length < r then
18:                collectionRow = new List<KVCell>()
19:                collection.push(collectionRow)
20:            else
21:                collectionRow = collection[r]
22:            end if
23:            collectionRow.push(next)
24:        end for
25:    end for
26:    this.Merge(this.cells, collection)
27: end function
```

7.2.2 ColorRects-Algorithmus

Für die Darstellung der farblichen Blöcke werden gefärbte Rechtecke ins Canvas gezeichnet. In den Mockups vom KV Diagramm (beispielsweise in Abbildung 4.11) geht der blaue farbliche Block über den Rand des KV Diagramms hinaus und teilt sich deshalb in zwei Rechtecke auf. Diese sind im Mockup mit spitzen Ecken versehen und haben vier Seiten. Die Darstellung solcher Blöcke erfolgt in den beiden vorgestellten Fremdtools aus Kapitel 2 *KVD - Karnaugh-Veitch-Diagramm* (s. Abschnitt 2.1.2) und *Karnaugh-Vetich-Map* (s. Abschnitt 2.2.2) hingegen mit abgerundeten Ecken und offenen Seiten. Diese Darstellung wird auch im Buch *Grundlagen der Technischen Informatik von Dirk. W. Hoffmann*[Hoffmann (2014)] und in der Veranstaltung *Theoretische Informatik bei Prof. Dr. Eisemann an der TH Köln* verwendet. Deshalb wird sie, anstelle von der in den Mockups, für dieses Lernmodul gewählt.
Es wurde ein Algorithmus geschrieben, der mehrere Rechtecke für einen Block erzeugt und bei jedem Rechteck die Information über fehlende Seiten des Rechtecks speichert. Der erste Teil des Algorithmus ist in Algorithmus 3 vorzufinden. Als Einstiegspunkt dient die Funktion *CreateColorRects*. Dieser wird der gezielte Block und die Feldgröße des KV Diagramms übergeben. Es wird die erste Zelle eines Blocks genommen und dessen X- und Y-Position auf dem KV Diagramm anhand Indexposition n bestimmt. Anschließend wird ein Objekt der Klasse *KVReflectingBlockRect* erstellt, auf die Größe des Blocks gebracht und der Liste *rects* hinzugefügt. Nach dem Durchlauf der Funktion *OpenRects* wird eine Liste mit allen Rechtecken vom Block zurückgeliefert. Innerhalb von *OpenRects* wird berechnet, wie weit das aktuelle Rechteck über den Rand hinausgeht. Wenn das Rechteck rechts und unten hinausragt, dann wird *OpenRectAll* aufgerufen. Bei nur einer hinausragenden Richtung wird dann für rechts *OpenRectRightLeft* und für unten *OpenRectUpDown* aufgerufen. Innerhalb dieser drei Funktionen werden dann weitere offene Rechtecke erzeugt. Die grafische Zellgröße beträgt 32x32 Pixel.

Algoritmus 3 ColorRects-Algorithmus

```
1:  function CreateColorRects(block, fieldWidth, fieldHeight)
2:      rects = new List<KVReflectingBlockRect>()
3:      cell = block.cells[0][0]
4:      x = (int)(cell.n % fieldWidth)
5:      y = (int)(cell.n / fieldWidth)
6:      rect = new KVReflectingBlockRect(x * 32, y * 32)
7:      rect.width = block.getWidth() * 32
8:      rect.height = block.getHeight() * 32
9:      rects.push(rect)
10:     return OpenRects(rects, fieldWidth * 32, fieldHeight * 32)
11: end function
```

Algoritmus 4 ColorRects-Algorithmus: OpenRects

```
1: function OpenRects(rects, fieldWidth, fieldHeight)
2:     rect = rects[0]
3:     xDistance = fieldWidth - (rect.x + rect.width)
4:     yDistance = fieldHeight - (rect.y + rect.height)
5:     if xDistance < 0 and yDistance < 0 then
6:         OpenRectAll(rect, rects, xDistance, yDistance)
7:     else if xDistance < 0 then
8:         OpenRectRightLeft(rect, rects, xDistance)
9:     else if yDistance < 0 then
10:        OpenRectUpDown(rect, rects, yDistance)
11:    end if
12:    return rects
13: end function
```

```
1: procedure OpenRectAll(r, rects, xD, yD)
2:     r.open.right = r.open.down = true
3:     rectUpLeft = new KVReflectingBlockRect(xD, yD, r.width, r.height)
4:     rectUpLeft.open.left = rectUpLeft.open.up = true
5:     rectUpRight = new KVReflectingBlockRect(r.x, yD, r.width, r.height)
6:     rectUpRight.open.right = rectUpRight.open.up = true
7:     rectDownLeft = new KVReflectingBlockRect(xD, r.y, r.width, r.height)
8:     rectDownLeft.open.left = rectDownLeft.open.down = true
9:     rects.push(rectUpLeft, rectUpRight, rectDownLeft)
10: end procedure
```

```
1: procedure OpenRectRightLeft(r, rects, xD)
2:     r.open.right = true
3:     newRect = new KVReflectingBlockRect(xD, r.y, r.width, r.height)
4:     newRect.open.left = true
5:     rects.push(newRect)
6: end procedure
```

```
1: procedure OpenRectUpDown(r, rects, yD)
2:     r.open.down = true
3:     newRect = new KVReflectingBlockRect(r.x, yD, r.width, r.height)
4:     newRect.open.up = true
5:     rects.push(newRect)
6: end procedure
```

7.3 Überprüfen der Ergebnisse

Nach Erstellung der Farbblöcke können dazugehörige Boolesche Ausdrücke eingegeben werden. Das Ergebnis soll auf seine Richtigkeit geprüft, ausgewertet und bei Bedarf angezeigt werden. Das gleiche gilt für DNF und KNF.
Die Reihenfolge innerhalb eines Ausdrucks ist nicht relevant, sofern das Endergebnis und die Form stimmt. Die Ergebnisprüfung wurde in die Klasse *KVExprCompare* ausgelagert (s. Tabelle 7.2). Zum Initialisieren dieser Klasse muss ein *BAExpression*-Objekt (s. Praxisprojekt, Kapitel 5.1.3) übergeben werden. Der Methode *equals* aus *KVExprCompare* kann dann ein weiteres Objekt übergeben werden. Für beide wird eine imaginäre Wahrheitstabelle aufgebaut. Sind die Ergebnisse identisch, so gelten die dazu gehörigen Booleschen Ausdrücke als äquivalent.
Passend zu jedem Farbblock wird ein korrekter Boolescher Ausdruck erzeugt und mit der Benutzereingabe verglichen. Bei den minimierten Termen DNF und KNF kommt jedoch noch vorher über Regex[2] eine Formprüfung hinzu.
Reguläre Ausdrücke (Regular Expression) ermöglichen ein sog. Pattern Matching d.h. es wird in einem Bereich nach einer gewissen Form gesucht. Die Zeichenkette wird mit Regex anhand einer bestimmten Zeichenabfolge durchsucht, die als sog. Regex-Satz festgelegt wird. Bei der Formprüfung vom DNF und KNF wurde der Regex-Satz für das Suchen von Klammern in einem Booleschen Ausdruck verwendet, der bereits im Praxisprojekt eingesetzt wurde (s. Praxisprojekt auf dem Datenträger, Kapitel 5.2). Um den Regex-Satz hier noch einmal hervor zu bringen wird der Screenshot aus dem Praxisprojekt in Abbildung 7.6 dargestellt. Dieser Reguläre Ausdruck sucht sich die Klammern inkl. Inhalt aus einem Booleschen Ausdruck und liefert diese als Menge zurück. Danach wird diese Menge anhand der Operatoren darauf geprüft, ob es sich hierbei um ein KNF oder DNF handeln könnte. Sind keine Klammern vorhanden, dann wird der Boolesche Ausdruck akzeptiert. Erst danach folgt der Vergleich über *KVExprCompare*.

Abbildung 7.6: Regextest von der Klammersuche [regex101]

[2]Regex: https://wiki.selfhtml.org/wiki/JavaScript/Objekte/RegExp

8 Vorstellung der Anwendung

In diesem Kapitel wird das Ergebnis der Implementierung in Form von Screenshots inklusive einer dazugehörige Beschreibung vorgestellt. Dabei wird auf die in Kapitel 5.2 aufgestellten funktionalen Anforderungen eingegangen.
Die nachfolgenden Screenshots wurden in einer iPhone 6 Simulation im Google Chrome Entwicklermodus erstellt. Die dort zu sehende Ansicht kann bei anderen Geräten optisch ein paar Unterschiede haben. Ein Vergleich von unterschiedlichen Geräten ist bei dem Gerätetest in Kapitel 9.1 vorzufinden.

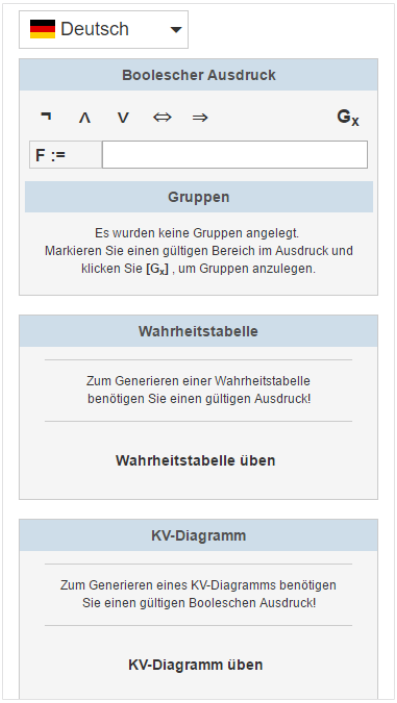


Abbildung 8.1: Startansicht

Weil eine Anforderung alternative Sprachen sind ist bei der Startansicht (s. Abbildung 8.1) ein Auswahlmenü für Sprachen eingesetzt worden. Um sicher zu sein, dass die Übersetzung korrekt ist wurden die Sprachen gewählt, die ich selbst beherrsche: Deutsch, Englisch und Russisch. Bei Änderung der Sprache schaltet die Applikation direkt auf die gewünschte Sprache um, ohne den Browser zu aktualisieren.
Die Startansicht wurde außerdem um einen neuen Bereich erweitert. In diesem Bereich befindet sich ein Informationstext und der Button *KV-Diagramm erstellen*. Beim Betätigen des Buttons wird, falls das Textfeld im Bereich *Boolescher Ausdruck* leer ist, dieses fokussiert. So wird der Benutzer dazu aufgefordert dort einen Booleschen Ausdruck einzugeben. Wurde ein gültiger Boolescher Ausdruck eingegeben kann ein KV Diagramm erstellt werden.

Abbildung 8.2: Generierung eines KV Diagramms

Die Applikation sollte bei Eingabe eines gültigen Booleschen Ausdrucks das dazugehörige KV Diagramm erzeugen.
In Abbildung 8.2 wurde der Boolesche Ausdruck F:=(A∧B)∧((C∨¬C)∧(¬D∨D))∨¬A∧¬B eingesetzt und das dazu passende KV Diagramm generiert. Der Ausdruck steht oben im Eingabefeld und es wurde ein 4x4-Feld erzeugt.
Die Besetzung der Variablen ist korrekt oberhalb und links neben KV Diagramm aufgereiht.

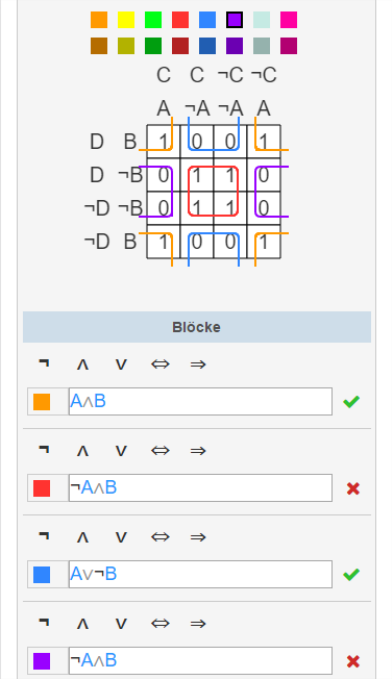


Abbildung 8.3: Farbliche Blöcke und deren Textfelder

Um dann den Booleschen Ausdruck aus Abbildung 8.2 zu minimieren, können Zellen mit Farben hinterlegt werden. Die Bildung der Blöcke ist von der Auswahl der Zellen abhängig.
In Abbildung 8.3 wurden alle Blöcke erzeugt, die für das Aufstellen des KNF und DNF wichtig sind. Für jeden Block erscheint dann auch ein Eingabefeld für Boolesche Ausdrücke. Die Zuordnung ist durch die Farben erkennbar. Die Booleschen Ausdrücke können durch Betätigen des *Ergebnis überprüfen* Buttons (s. Abbildung 8.4) auf ihre Richtigkeit geprüft werden. Der Ausdruck im roten und violetten Block ist falsch und erhält deshalb ein rotes Kreuz anstelle eines grünen Hakens.
Die Fehler könnten unterschiedliche Ursachen haben. Beim roten Ausdruck müsste das *B* negiert und beim violetten Ausdruck müsste ein *Oder* anstelle des *Und*.

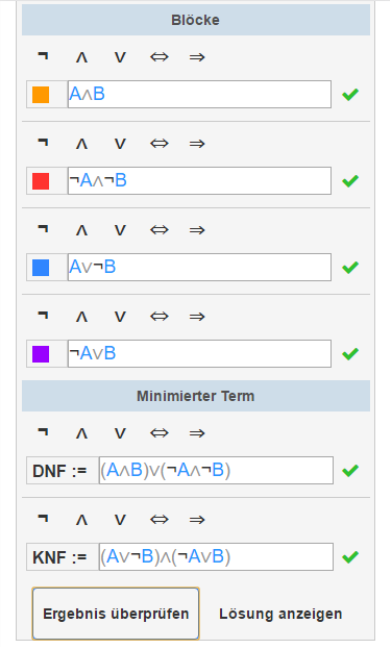


Abbildung 8.4: Überprüfung der Lösung

Abbildung 8.4 zeigt, wie die Booleschen Ausdrücke in Abbildung 8.3 vollständig richtig aussehen. Außerdem sind hier die Felder für das KNF und DNF befüllt und ausgewertet.
Die Endergebnisprüfung wurde bereits in Kapitel 7.3 erläutert.

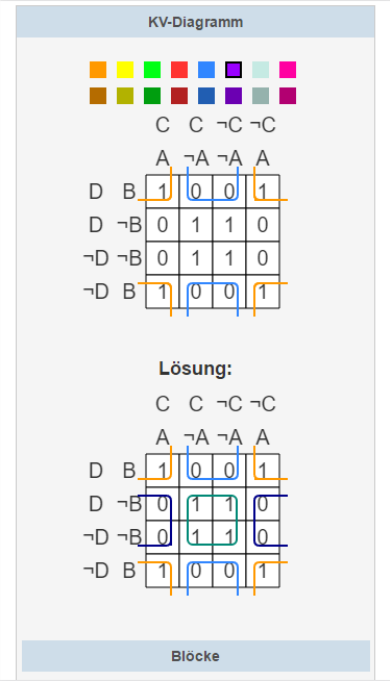


Abbildung 8.5: Lösung vom KV Diagramm anzeigen

Wenn der Benutzer die Lösung nicht kennt, kann er sich diese mit dem Button *Lösung anzeigen* (s. Abbildung 8.4) anzeigen lassen.
Das Anzeigen der Lösung wird in Abbildung 8.5 und 8.6 dargestellt.
In Abbildung 8.5 wird die Lösung für das KV Diagramm angezeigt. Korrekte Blöcke werden zusammen mit ihrer Farbe übernommen.
Fehlende Blöcke werden mit einer zufälligen Farbe ergänzt.
Für Blockbildung wird ebenfalls der Algorithmus 1 verwendet.

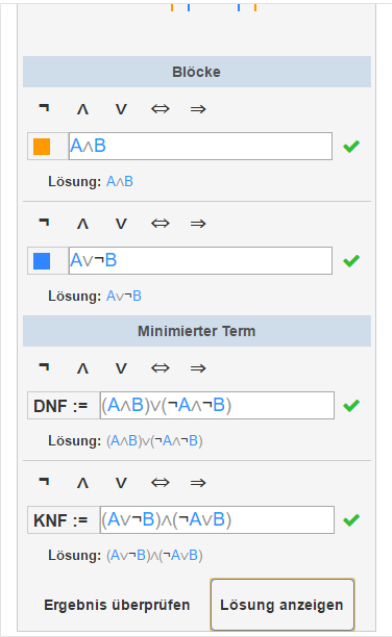


Abbildung 8.6: Lösung der Eingaben anzeigen

In Abbildung 8.6 werden Lösungen zu den Booleschen Ausdrücken unter dem Textfeld angezeigt. Diese können bei Bedarf markiert und kopiert werden.

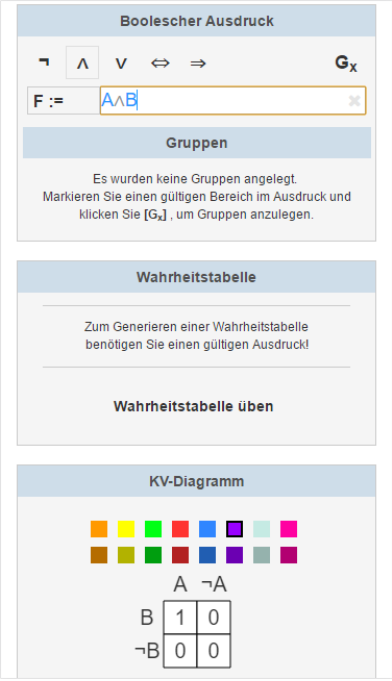


Abbildung 8.7: Automatische Neuerstellung

Die Änderung des Booleschen Ausdrucks aktualisiert das KV Diagramm passend zum Ausdruck und entfernt alle farblichen Blöcke. Da die Eingabefelder für farbliche Blöcke an diese gebunden sind, werden sie automatisch mit entfernt.

9 Evaluation

In diesem Kapitel findet eine Leistungsbewertung der Applikation statt.
Die Applikation wurde auf unterschiedlichen Browsern und Geräten getestet und das Ergebnis über Screenshots in Abschnitt 9.1 festgehalten. Zusätzlich wurde eine Umfrage für die in Kapitel 5.1.3 aufgeführte Zielgruppe erstellt und in Abschnitt 9.2 ausgewertet.

9.1 Lauffähigkeit auf unterschiedlichen Plattformen

In Kapitel 5.3.6 wurde die Wichtigkeit für die Lauffähigkeit auf dem Browser Google Chrome und Firefox betont. Der Vollständigkeit halber wird die Applikation auch auf Apple Geräten im Safari Browser getestet. Computer und Laptops werden in Abschnitt 9.1.1 zusammen betrachtet und mobile Endgeräte erhalten ein eigenes Unterkapitel (s. Abschnitt 9.1.2).

9.1.1 Computer und Laptop

In diesem Abschnitt wird die Applikation auf einem Computer in der Browser-Fenstergröße von 1024x768 Pixel gestartet, um das komplette Fenster zu befüllen. Auf einem Laptop würde das Ergebnis dasselbe sein.

Google Chrome

Bei Abbildung 9.1 handelt es sich um eine Google Chrome Version 56.0.2924.87 auf einem Windows 10 Computer. Das Ergebnis ist genau so, wie es gewünscht ist, da dieser Browser bei der Entwicklung verwendet wurde.

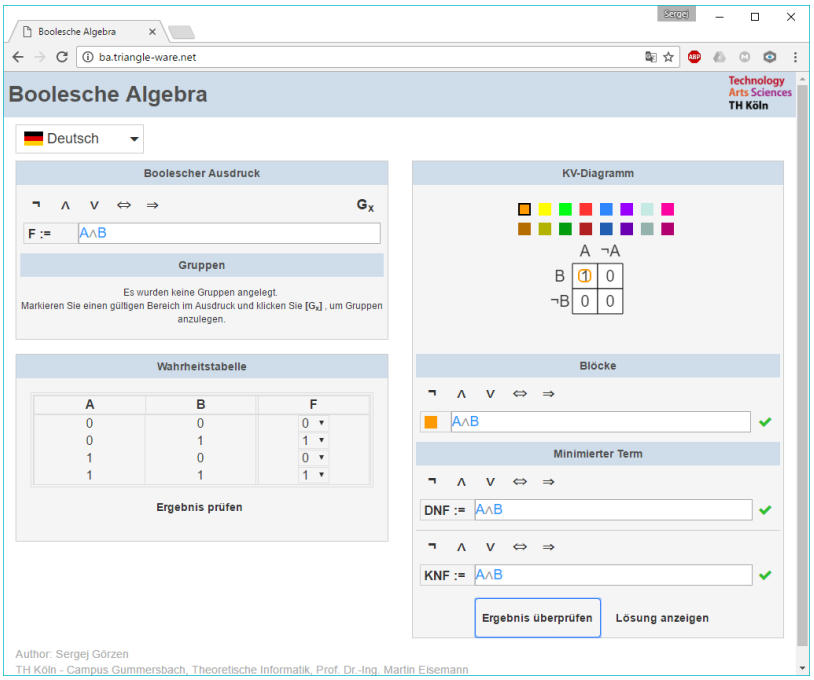


Abbildung 9.1: Evaluation auf Windows mit Google Chrome

Mozilla Firefox

Für Abbildung 9.2 wurde Mozilla Firefox mit der Version 51.0.1 verwendet. Bei diesem Browser gibt es ebenfalls nichts zu bemängeln.

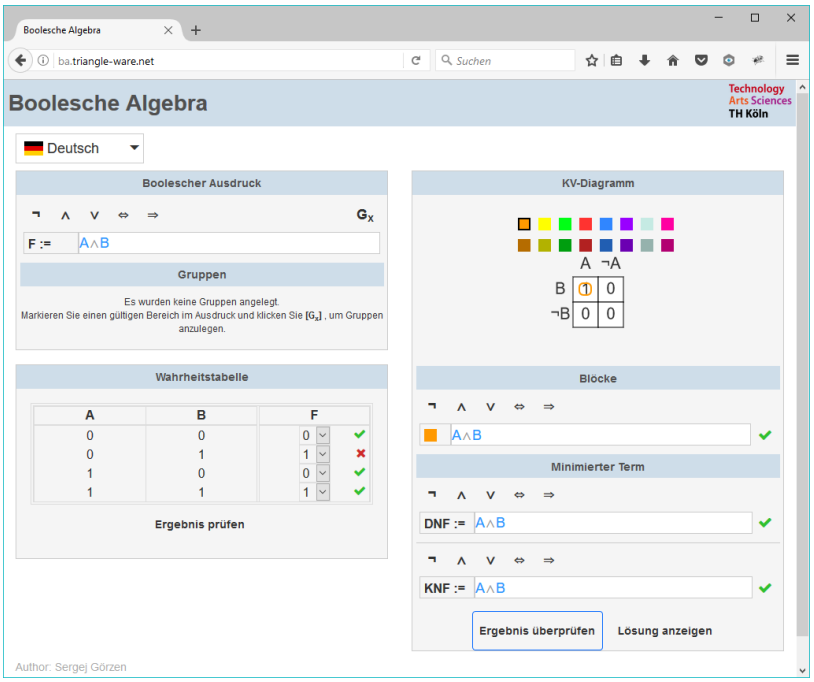


Abbildung 9.2: Evaluation auf Windows mit Firefox

Safari

Für den Test mit Safari wurde das Betriebssystem Mac OS X Yosemite Version 10.10.5 mit Safari 8 verwendet. Da bei den Auswahlboxen in der Wahrheitstabelle das Standard HTML Element *Select* verwendet wird, setzt jeder Browser seinen eigenen Stil hin. Je nach Fenstergröße könnten die Zahlen dann abgeschnitten werden (wie in Abbildung 9.3). Der einzige weitere Unterschied ist, dass einige Symbole, wie zum Beispiel die Operatoren etwas anders, als in Google Chrome (s. Abbildung 9.1) aussehen.

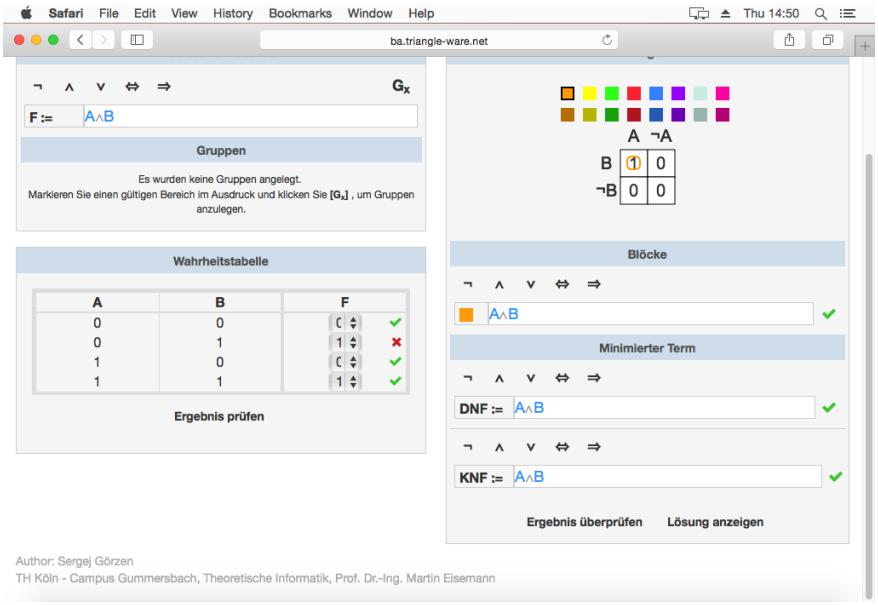


Abbildung 9.3: Evaluation auf Mac OS X Yosemite mit Safari

9.1.2 Mobile Endgeräte

Da es sich um eine Webapplikation handelt ist es wichtig zu wissen, wie die Applikation auf echten mobilen Endgeräten aussieht. Dieser Abschnitt zeigt Screenshots und Auswertung des Tests.

Android

Es wurde ein Sony Xperia Z3+ mit der Android Version 6.0.1 verwendet.
Es kamen der Google Chrome und Mozilla Firefox Browser zum Einsatz.
In Abbildung 9.4 und 9.5 sind die Screenshot des Tests mit Google Chrome auf einem Android Gerät zu beobachten. Es sieht alles, wie gewünscht und erwartet aus. Bei der Firefox Version ist ebenfalls nichts zu bemängeln (s. Abbildung 9.6 und 9.7).

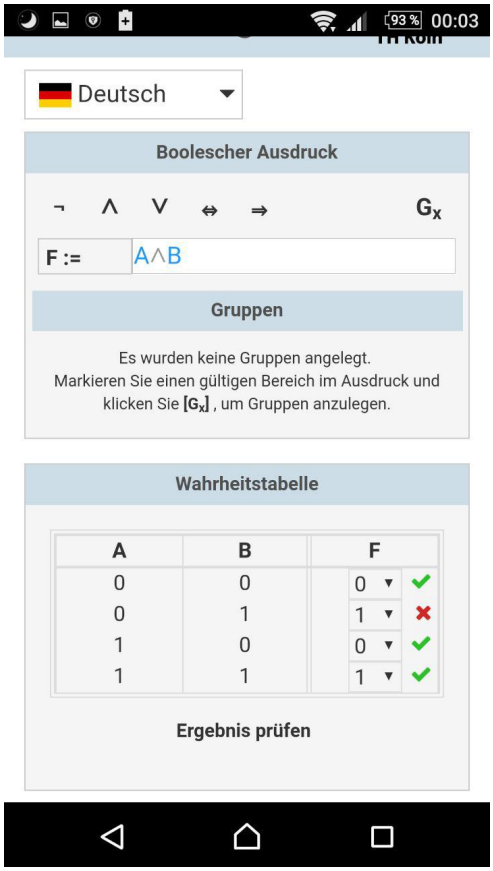


Abbildung 9.4: Evaluation auf Android mit Google Chrome 1

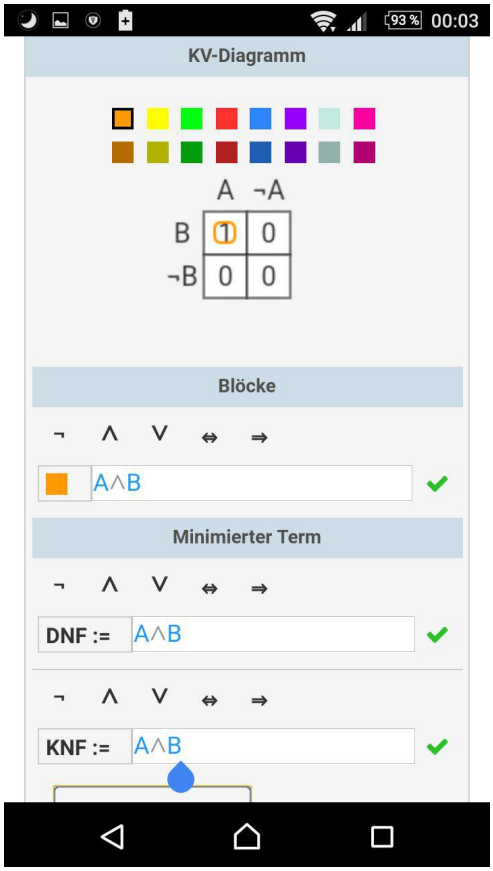


Abbildung 9.5: Evaluation auf Android mit Google Chrome 2

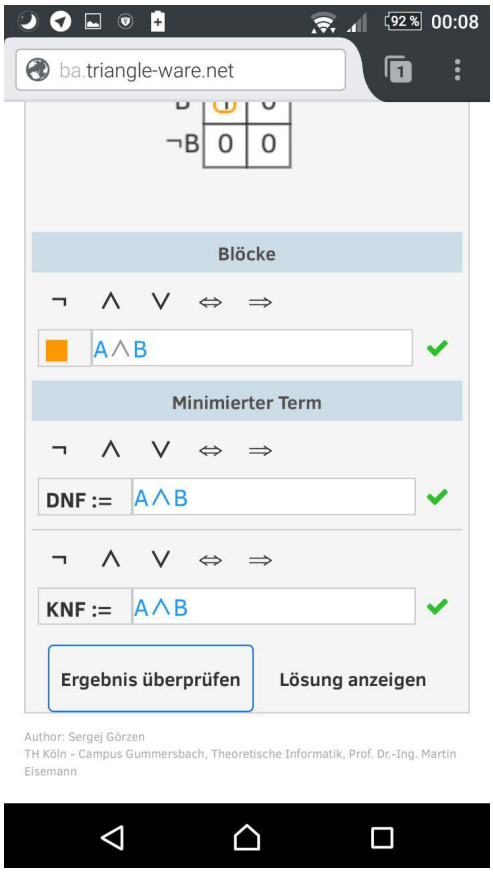


Abbildung 9.6: Evaluation auf Android mit Firefox 1

Abbildung 9.7: Evaluation auf Android mit Firefox 2

iOS

Als Einsatzgerät wurde ein iPhone 5 mit iOS 10.0.1 (14A403) mit dem Safari Browser verwendet.

Gegensatz zu Android gibt es hier einige optische Mängel. Genau, wie bereits bei Mac OS X werden beim Safari eigene Auswahlboxen verwendet, wodurch die 0en und 1en bei der Wahrheitstabelle optisch verschluckt werden. Außerdem sind die Auswahlfelder für Farben nicht quadratisch. Weitere Mängel sind nicht bekannt.

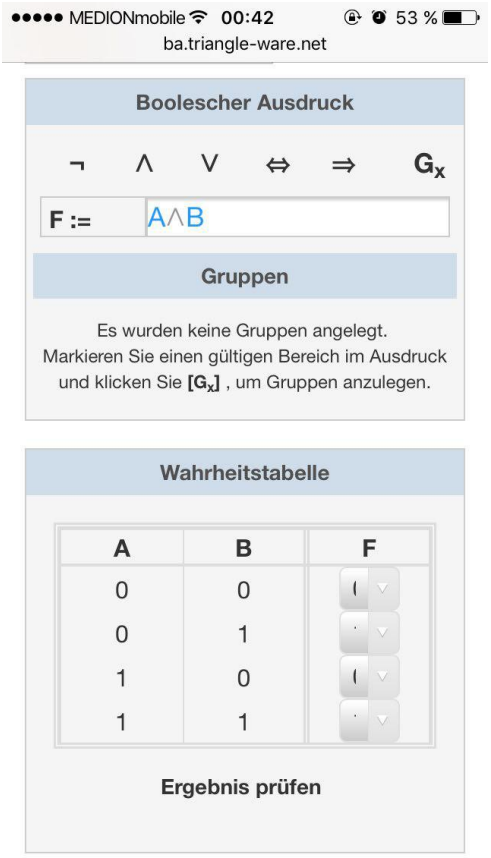


Abbildung 9.8: Evaluation auf iPhone mit Safari 1

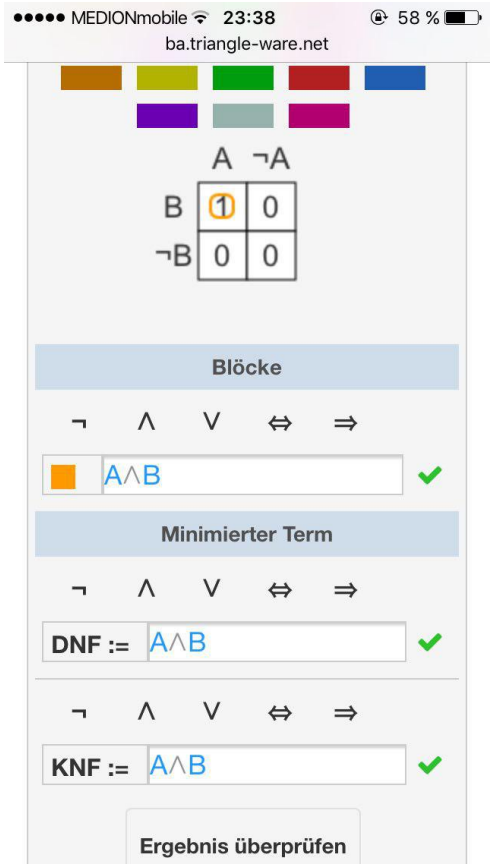


Abbildung 9.9: Evaluation auf iPhone mit Safari 2

9.2 Benutzerumfrage

Für das Projekt wurde mit Google Forms[1] eine Nutzerumfrage für den Zeitraum vom 30.01.2017 bis 07.02.2017 erstellt. Durch die Umfrage sollten Fehler entdeckt und Verbesserungsvorschläge entgegen genommen werden. Dieses Kapitel beschäftigt sich mit der Auswertung dieser Umfrage. Auf dem CD Datenträger befinden sich zwei von Google generierten PDF-Dateien für das Ergebnis. Obwohl es bei der Umfrage nur vier Teilnehmer gab, kann ein eindeutiges Ergebnis festgehalten werden.
Die Nutzer sollten bei der Umfrage drei Aufgaben bearbeiten und Fragen zum Umgang beantworten. Einer der Befragenden hat eine Lücke bei der Überprüfung der Ergebnisse gefunden: das KNF wurde im Eingabefeld des DNF als korrekte Lösung akzeptiert (und ebenfalls anders herum). Dies lag daran, dass es keine Formprüfung vor Validierung des Ergebnisses gab. Dieses Problem wurde in Kapitel 7.3 mit dem Einsatz von Regex beseitigt. Ein weiterer Befragende hat als Verbesserungsvorschlag geäußert, dass ein Button für das automatische Einsetzen von Klammern nützlich wäre. Da diese Funktion noch nicht abgesprochen wurde und nur einen Eintrag in der Direktive *boolExpr* erfordert, wurde dieser auskommentiert eingetragen und kann durch das Entfernen des Kommentars implementiert werden.
Insgesamt war laut der Auswertung das Erlebnis der Nutzung sehr zufriedenstellend. Es traten grundsätzlich keine technischen Schwierigkeiten auf, die in Zusammenhang mit dem Gerät oder Browser gebracht werden konnten. Da bei allen Nutzern Windows und Google Chrome zum Einsatz kam, konnte die Fehlerrate bei unterschiedlichen Plattformen eher weniger geprüft werden. Von daher wäre bei einer Weiterentwicklung eine fortführende Evaluation für Apple und Android Geräte auf längere Dauer besonders wichtig, um Fehlern entgegen wirken zu können.

[1] Google Forms: `https://www.google.de/intl/de/forms/about/`

10 Selbstreflexion

Die Selbstreflexion hilft dabei den Ablauf des Projekts zu bewerten, Schwierigkeiten zu beurteilen und ein Fazit zu festzuhalten.

10.1 Ablauf

Der erste Meilenstein bestand darin, Ideen darüber aufzustellen, was und wie umgesetzt werden soll. Dabei war es wichtig sich bereits vorhandene Applikationen (s. Kapitel 2) anzusehen und auszuprobieren.
Im zweiten Meilenstein wurde das Ziel gesetzt weitere Sprachen zu implementieren und das KV Diagramm korrekt anzuzeigen. Für das Anzeigen des KV Diagramms wurden einige HTML5 Canvas Frameworks durchprobiert und das nützlichste rausgesucht (s. Kapitel 6.5).
Der dritte Meilenstein beinhaltete die Konzeption und Implementierung des Such-Algorithmus (s. Algorithmus 1).
Im vierten Meilenstein wurde der Such-Algorithmus und das HTML5 Canvas Framework dafür benutzt, um die farblichen Blöcke anzuzeigen und die Eingabefelder für Booleschen Ausdrücke der Blöcke zu implementieren. Der letzte Meilenstein beschäftigte sich mit Beseitigen von Bugs und dem Testen der Applikation auf unterschiedlichen Geräten.

10.2 Schwierigkeiten

Bei der Konzeption und Umsetzung gab einige wenige Hürden zu bewältigen. In diesem Abschnitt werden daher auftretende Probleme, sowie deren Lösungsansatz aufgeführt.

10.2.1 Such-Algorithmus

Die größte Hürde entstand beim Such-Algorithmus, da bei der Konzeption ein Denkfehler unterlaufen war. Dieser wurde vorerst auch umgesetzt, sodass ein nicht vollständig funktionierendes und träges Ergebnis entstand. Deshalb wurde das gesamte Konzept verworfen und ein neues erstellt, welches in Kapitel 4.4.3 erläutert wurde.

10.2.2 Optische Ungleichheiten

Auf unterschiedlichen Browsern sah es teilweise nicht so, wie geplant, aus.
Die meisten Schwierigkeiten wurden durch Css-Fallback[1] Techniken beseitigt. Besonders betroffen waren neuere CSS Funktionen, wie *Transition*[2], *Transform*[3], *@keyframes*[4] und *calc*[5].
Solange die Funktionalität nicht beeinträchtigt ist, fallen einige Unterschiede kaum auf und müssen eher sekundär behandelt werden.
Der Bedarf an Änderungen ist eindeutig bei den Auswahlboxen vorhanden. Für eine globale Lösung müsste ein einheitlicher CSS-Stil für eine *Select*-Box gewählt werden und entwickelt werden. Der Standardstil müsste dann überschrieben werden.

10.3 Fazit

In der Anforderungsanalyse wurden in Kapitel 5.2 funktionale Anforderungen aufgestellt. Diese werden nun in Tabelle 10.1 noch einmal (in Kurzfassung) aufgezählt und daraufhin untersucht, ob diese erfüllt wurden.

Kurzbeschreibung	**Status**
Generierung eines KV Diagramm zum gültigen Booleschen Ausdruck	Erledigt
Zusammenstellung von farblichen Blöcken durch Klicken auf KV Diagramm Zellen	Erledigt
Eingabefelder für farbliche Blöcke	Erledigt
Eingabefelder für KNF und DNF	Erledigt
Validierung der Eingaben	Erledigt
Anzeigen der Lösungen auf Anfrage	Erledigt
Automatische Anpassung des KV Diagramms bei Änderung des Booleschen Ausdrucks	Erledigt
Sprachauswahl	Erledigt

Tabelle 10.1: Checkliste der funktionalen Anforderungen

Grundsätzlich funktioniert die Applikation auf jedem Gerät. Optische Feinheiten müssen jedoch vorgenommen werden.
Zusammenfassend lässt sich sagen, dass alle geforderten Funktionalitäten umgesetzt

[1] CSS-Fallback: `https://css-tricks.com/quickie-css3-tricks-with-fallbacks/`
[2] CSS-Transition: `http://www.w3schools.com/css/css3_transitions.asp`
[3] CSS-Transform: `http://www.w3schools.com/cssref/css3_pr_transform.asp`
[4] CSS-Keyframes: `http://www.w3schools.com/cssref/css3_pr_animation-keyframes.asp`
[5] CSS-Calc: `http://www.w3schools.com/cssref/func_calc.asp`

wurden. Die Applikationen kann nun auch zum Üben von Minimierung mit Hilfe des KV Diagramm verwendet werden.

10.4 Ausblick

Da die Umsetzung der Schaltelemente den Rahmen dieser Arbeit gesprengt hätte, wäre dessen Umsetzung der nächste große Schritt, um die Applikation zu erweitern. Einige Gedanken dazu wurden bereits in Kapitel 4.5 geäußert. Außerdem könnte ein besseres Design konzipiert und umgesetzt werden.
Zu beachten sind auch die Vorschläge aus dem Praxisprojekt (Praxisprojekt, s. Kapitel 7.4).

Glossar

Algorithmus	Eine eindeutige Handlungsvorschrift zur Lösung eines Problems.
Bitnummerierung	Alle möglichen Kombinationen von Variablen. Sind zum Beispiel A, B und C (n=3) gegeben, so werden alle Kombinationen aus 0 und 1 genommen. Es sind immer 2^n Kombinationen möglich.
Canvas	Canvas ist eine Leinwand in der Sprache HTML, auf der grafische Objekte dargestellt werden können.
Disjunktion	Eine logische Oder-Verknüpfung zwischen zwei Aussagen.
Div	Ein nahezu regelnloses HTML-Tag. Wird in der Regel als unsichtbarer Wrapper oder Box benutzt.
DOM/DOM-Baum	Document Object Model ist die HTML-Struktur als Schnittstelle in Form eines Objekts.
Eingabeaufforderung	Eingabefenster für die Eingabe von Systembefehlen. (In Linux ist es das Terminal).
Fallback-Technik	Sollte eine Ausführung nicht klappen, so greifen stufenweise Ersatzausführungen.
farblicher Block	Sammlung an Zellen im KV Diagramm, welche an eine Farbe gebunden sind und somit einen gefärbten Block bilden.
HTML-Tag	Ein HTML-Knoten, mit hinterlegter Bedeutung und Funktion. Beispiel: Div, Span, Img, Sub
Konjunktion	Eine logische Und-Verknüpfung zwischen zwei Aussagen.
Livereload	Die automatische Aktualisierung des Quellcodes im Browser. Bei einer Änderung des Quellcodes in der IDE, lädt der Browser neuste Änderungen, ohne Zusatzaufforderung, nach.
Modularität	Aufbau nach einem Baukastensystem, wo einzelne Elemente unabhängig funktionieren und ohne Mehraufwand ins System integriert werden können.
Pattern Matching	Eine musterbasierte Suche für eine diskrete Struktur.

Regex ..... Ein regulärer Ausdruck (Regular Expression) ist in der Theoretischen Informatik eine Zeichenkette, die der Beschreibung von bestimmter Syntax dient.

Responsive Design .. Ein Design, dass sich jeder Bildschirmgröße eines Gerätes anpasst.

Root-Verzeichnis ... Höchstes Verzeichnis eines Ordners.

Shapes ..... Eine Form oder Kontur eines Objekts.

Startansicht ..... Ansicht, die beim Start einer Applikation erscheint.

SVG ..... Steht für Scalable Vector Graphics und repräsentiert auf vektorenbasierte Grafiken.

Teilausdruck ..... Ein kleiner Ausschnitt des Ausdrucks, der als eigener Ausdruck existieren kann.

Usability ..... Das Ausmaß, in dem ein Produkt durch bestimmte Benutzer in einem Nutzungskontext benutzt werden kann, um ein Ziel effektiv, effizient und zufriedenstellend zu erreichen.

Variable ..... Ein Zeichen, das als Platzhalter für Zahlen dient.

Quellenverzeichnis

[balsamiq] BALSAMIQ: *Balsamiq. Rapid, effective and fun wireframing software.* https://balsamiq.com/

[Bächle 1996] BÄCHLE, Michael: *Qualitätsmanagement der Softwareentwicklung.* 1996

[eLearning Boolean Alg] BOOLEAN ALG eLearning: *eLearning Modul für Boolesche Algebra.* http://www.gm.fh-koeln.de/~eisemann/eLearning/TI/Boole/

[Daniel Kuhn 2013] DANIEL KUHN, Miachael R.: *Performante Webanwendungen: Client- und serverseitige Techniken zur Performance-Optimierung.* 2013

[dpa 2011] DPA: *Lerntechniken: E-Learning-Angebote effektiv nutzen.* https://www.berlin.de/special/jobs-und-ausbildung/uni-und-studium/studentenleben/1265956-1018135-lerntechnikenelearningangeboteeffektivnu.html. Version: 2011

[draw.io] DRAW.IO: *Flowchart Maker & Online Diagram Software.* https://www.draw.io/

[E-Bude] E-BUDE: *Elektroniker Bude.* http://www.elektroniker-bu.de/

[GooglePlay a] GOOGLEPLAY: *Boole von José Antonio Riaza Valverde.* https://play.google.com/store/apps/details?id=boole.riazavalverde.com&rdid=boole.riazavalverde.com. Version: a

[GooglePlay b] GOOGLEPLAY: *KVD - Karnaugh-Veitch-Diagram von cits.* https://play.google.com/store/apps/details?id=com.mhsoft.kvd&hl=de. Version: (b)

[Hoffmann 2014] HOFFMANN, Dirk W.: *Grundlagen der Technischen Informatik.* 2014

[Hoffmann 2015] HOFFMANN, Dirk W.: *Theoretische Informatik.* 2015

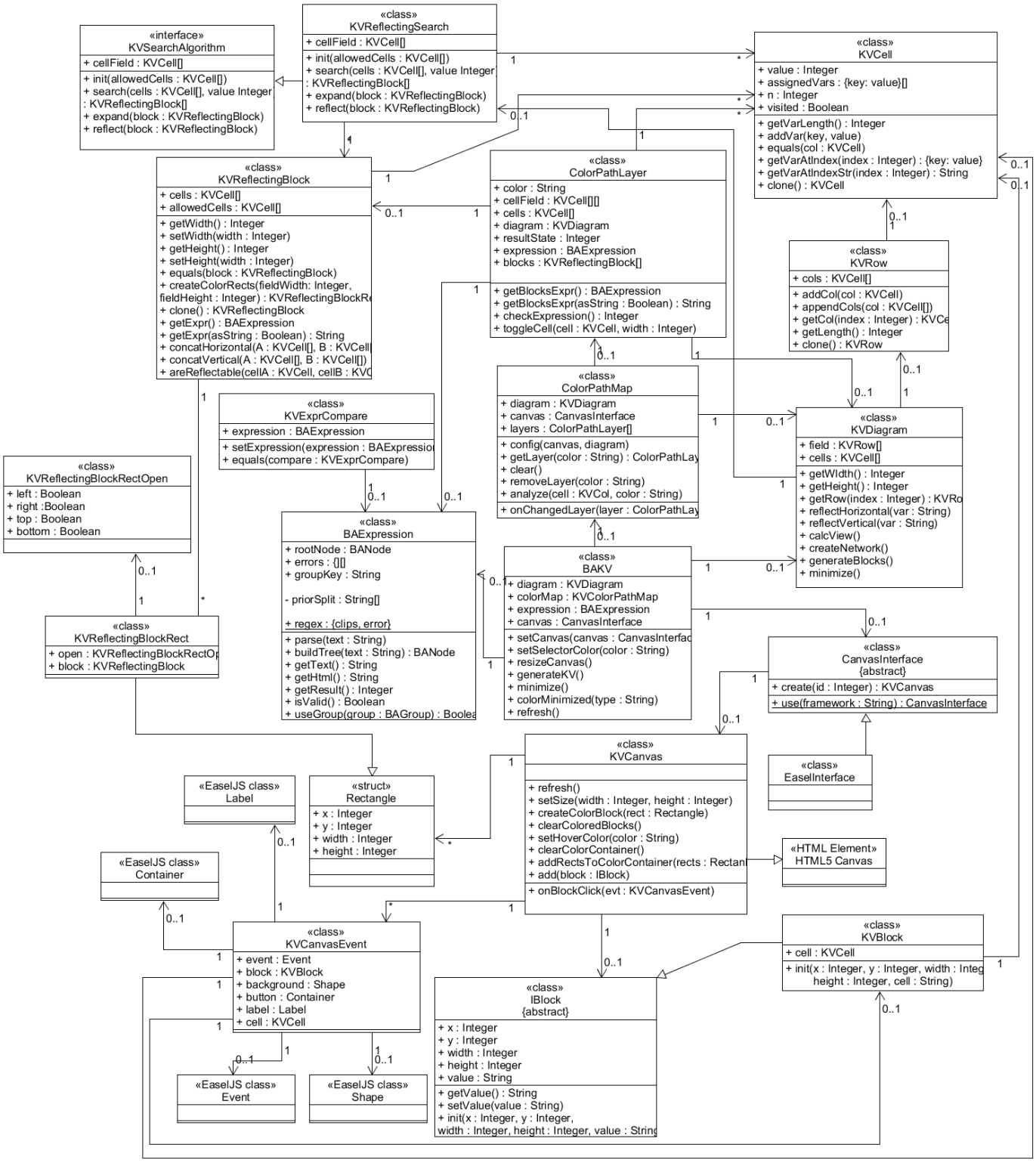
[Marburg] MARBURG: *Karnaugh Veitch Map Uni Marburg.* http://www.mathematik.uni-marburg.de/~thormae/lectures/ti1/code/karnaughmap/

[regex101] REGEX101: *Online regex tester.* https://regex101.com/

[umlet.com] UMLET.COM: *UMLet - Free UML Tool.* http://www.umlet.com/

Anhang

Klassendiagramm - Großansicht



Eidesstattliche Erklärung

Ich versichere, die von mir vorgelegte Arbeit selbständig verfasst zu haben.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben.

Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Gummersbach, 7. Februar 2017

Sergej Görzen